

Escuela Politécnica Superior

20
21

Trabajo fin de máster

Exploración dinámica en recomendación con bandidos multi-brazo



Javier Aróstegui Martín

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C/ Francisco Tomás y Valiente nº 11

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR



Máster en Investigación e Innovación en Inteligencia Computacional y Sistemas Interactivos

TRABAJO FIN DE MÁSTER

**Exploración dinámica en recomendación con
bandidos multi-brazo**

Autor: Javier Aróstegui Martín
Tutor: Pablo Castells Azpilicueta

julio 2021

Todos los derechos reservados.

Queda prohibida, salvo excepción prevista en la Ley, cualquier forma de reproducción, distribución comunicación pública y transformación de esta obra sin contar con la autorización de los titulares de la propiedad intelectual.

La infracción de los derechos mencionados puede ser constitutiva de delito contra la propiedad intelectual (*arts. 270 y sgts. del Código Penal*).

DERECHOS RESERVADOS

© 3 de Noviembre de 2017 por UNIVERSIDAD AUTÓNOMA DE MADRID
Francisco Tomás y Valiente, nº 1
Madrid, 28049
Spain

Javier Aróstegui Martín

Exploración dinámica en recomendación con bandidos multi-brazo

Javier Aróstegui Martín

C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

RESUMEN

Los sistemas de recomendación se han convertido en un pilar importante en servicios *online* en los últimos años, pero los que se utilizan hoy día aún tienen margen de mejora. Existen distintos caminos por los que investigar para intentar mejorar estos sistemas. Uno de esos caminos es la utilización del aprendizaje por refuerzo para la recomendación.

El aprendizaje por refuerzo intuitivamente se adapta mucho mejor a la tarea de recomendación. Una de las principales aproximaciones de aprendizaje por refuerzo que se investigan para aplicar a la recomendación son los bandidos multibrazo. Los bandidos multibrazo se enfrentan a un dilema llamado el dilema de la exploración-explotación que consiste en saber cuándo el algoritmo debe maximizar la recompensa a corto plazo o buscar otra acción que le lleve a recompensas mejores a largo plazo.

Para resolver este dilema existen distintas estrategias que pueden variar de manera dinámica. La exploración dinámica se encarga de ajustar cuanto y cuando se debe explorar y cuanto y cuando explotar a lo largo de la ejecución de tal manera que el agente puede modificar su comportamiento según la situación en la que se encuentre. En este trabajo se plantean y prueban cuatro algoritmos que hacen uso de la exploración dinámica para comparar con otros tipos de bandidos multibrazo y técnicas utilizadas hoy día de filtrado colaborativo como *baseline*.

He explorado nuevas variantes de exploración dinámica, he implementado tres de estos algoritmos (añadiendo un hiperparámetro original en uno de ellos) y he creado una variante original. Con esto se han realizado una serie de experimentos que nos muestran su rendimiento en distintas métricas de novedad y diversidad además de acierto y tiempo de ejecución. Cualquiera de los algoritmos de exploración dinámica puede comportarse de manera estática si los hiperparámetros son correctos. Es por esto por lo que cualquiera de las variaciones de exploración dinámica obtiene un acierto mayor que a su contra parte estática (siempre y cuando se haga una buena optimización de hiperparámetros y sea el acierto la métrica a maximizar). El comportamiento general de estos algoritmos dinámicos son prometedores ya que en algunos casos rinde mejor en todas las métricas respecto a las variantes estáticas. Como contenido original de este trabajo se ha propuesto el algoritmo AdaptiveUCB (variante del algoritmo UCB que modula la importancia del término exploratorio a lo largo de la ejecución) y un nuevo hiperparámetro para Dynamic Thompson Sampling que le permite tardar aproximadamente diez veces menos en ejecutarse obteniendo un acierto y una diversidad mayor al original.

PALABRAS CLAVE

recomendación, bandidos multi-brazo, diversidad, novedad, exploración, explotación

ABSTRACT

Recommender systems have become an important pillar in online services in recent years, but those used today still have room for improvement. There are different lines of investigation trying to improve these systems. One of those lines is to use reinforcement learning for recommendation.

Reinforcement learning is intuitively much better suited for the recommendation task. One of the main reinforcement learning approaches being investigated is multi-armed bandits. Multi-armed bandits face a dilemma called the exploration-exploitation dilemma, which is whether the algorithm should maximize the payoff in the short term or look for other actions that might lead to better rewards in the long term.

To solve this dilemma there are different strategies that treat it dynamically. The dynamic exploration adjusts how much and when to explore and how much and when to exploit throughout the execution in such a way that the agent can modify its behavior according to the situation. In this work, four algorithms that make use of dynamic exploration are proposed and tested comparing them to other types of multi-armed bandits and to other collaborative filtering techniques used today as baseline.

I have explored new variants of dynamic exploration, implemented three of these algorithms (adding an original hyperparameter in one of them), and created an original variant. With this, a series of experiments have been carried out that show its performance in different metrics of novelty and diversity as well as accuracy and execution time. Any of the dynamic exploration algorithms can behave statically if the hyperparameters are correct. This is why any of the dynamic exploration variations obtains a higher accuracy than its static counterpart (as long as a good hyperparameter optimization is done and the accuracy is the metric to be maximized). The general behavior of these dynamic algorithms is promising as in some cases it performs better than the static variants in all metrics. As original content of this work it has been proposed the algorithm `textit AdaptiveUCB` (a variant of the UCB algorithm that modulates the importance of the exploratory term throughout the execution) and a new hyperparameter for `textit Dynamic Thompson Sampling` which makes the algorithm ten times faster, obtaining a success and greater diversity than the original.

KEYWORDS

recommendation, multi-armed bandits, novelty, diversity, exploration, exploitation

ÍNDICE

| | | |
|----------|---|-----------|
| 1 | Introducción | 1 |
| 1.1 | Motivación | 1 |
| 1.2 | Objetivos | 2 |
| 1.3 | Estructura del documento | 2 |
| 2 | Estado del arte | 5 |
| 2.1 | Sistemas de recomendación | 5 |
| 2.1.1 | Tarea de recomendación | 5 |
| 2.1.2 | Objetivos de la recomendación | 6 |
| 2.1.3 | Técnicas de recomendación | 7 |
| 2.2 | Aprendizaje por refuerzo | 10 |
| 2.2.1 | Formulación del problema | 10 |
| 2.2.2 | Exploración dinámica en bandidos multibrazo | 11 |
| 3 | Diseño y desarrollo | 13 |
| 3.1 | Implementación y planteamiento | 13 |
| 3.2 | Análisis de los datos | 13 |
| 3.3 | Bandidos multi-brazo con exploración dinámica | 14 |
| 3.3.1 | Adaptive ϵ -greedy | 14 |
| 3.3.2 | VDBE ϵ -greedy | 15 |
| 3.3.3 | ϵ -first Greedy | 17 |
| 3.3.4 | Dynamic Thompson Sampling | 17 |
| 3.3.5 | Adaptive UCB | 19 |
| 3.4 | Métricas de novedad y diversidad | 20 |
| 4 | Experimentos y resultados | 23 |
| 4.1 | Análisis de hiperparámetros | 23 |
| 4.1.1 | Adaptive ϵ -greedy | 23 |
| 4.1.2 | VDBE ϵ -greedy | 24 |
| 4.1.3 | ϵ -first Greedy | 24 |
| 4.1.4 | Dynamic Thompson Sampling | 26 |
| 4.2 | Comparativa de algoritmos | 28 |
| 4.2.1 | Comparativa de algoritmos basados en ϵ -greedy | 30 |
| 4.2.2 | Comparativa de algoritmos basados en UCB | 30 |

| | |
|--|-----------|
| 4.2.3 Comparativa de algoritmos basados en Thompson Sampling | 32 |
| 4.2.4 Comparativa de otros algoritmos | 32 |
| 4.2.5 Comparativa final de algoritmos | 33 |
| 5 Conclusiones y trabajo futuro | 37 |
| 5.1 Conclusión | 37 |
| 5.2 Trabajo futuro | 38 |
| Bibliografía | 41 |

LISTAS

Lista de algoritmos

| | | |
|-----|---|----|
| 3.1 | Pseudocódigo de Adaptive ϵ -greedy. | 15 |
| 3.2 | Pseudocódigo de VDBE ϵ -greedy. | 17 |
| 3.3 | Pseudocódigo de ϵ -first greedy. | 17 |
| 3.4 | Pseudocódigo de parameterUpdate. | 19 |
| 3.5 | Pseudocódigo de Dynamic Thompson Sampling. | 19 |
| 3.6 | Pseudocódigo de AdaptiveUCB. | 20 |

Lista de ecuaciones

| | | |
|-----|--|----|
| 3.1 | Fórmula de cálculo de la función para actualizar ϵ en VDBE ϵ -greedy. | 16 |
| 3.2 | Fórmula de actualización de ϵ en VDBE ϵ -greedy. | 16 |
| 3.3 | Ecuación para elegir ítem en Thompson Sampling. | 18 |
| 3.4 | Ecuación para elegir ítem en UCB. | 19 |
| 3.5 | Término exploratorio en UCB. | 20 |
| 3.6 | Calculo γ en AdaptiveUCB. | 20 |

Lista de figuras

| | | |
|-----|---|----|
| 2.1 | Estructura de datos de <i>ratings</i> | 6 |
| 2.2 | Esquema user-based knn. | 9 |
| 2.3 | Esquema del aprendizaje por refuerzo. | 10 |
| 3.1 | Evolución del parámetro ϵ en Adaptive ϵ -greedy. | 14 |
| 3.2 | Evolución del parámetro ϵ en VDBE ϵ -greedy. | 16 |
| 3.3 | Evolución de la distribución Beta según α y β aumentan. | 18 |
| 3.4 | Evolución del parámetro γ en AdaptiveUCB. | 21 |
| 4.1 | Acierto frente al parámetro f en Adaptive ϵ -greedy. | 24 |
| 4.2 | Acierto frente al parámetro <i>limit</i> en Adaptive ϵ -greedy. | 25 |
| 4.3 | Acierto frente al parámetro δ en VDBE ϵ -greedy. | 25 |
| 4.4 | Acierto frente al parámetro σ en VDBE ϵ -greedy. | 26 |

| | | |
|------|--|----|
| 4.5 | [Acierto frente al parámetro <i>iterations</i> en ϵ first-greedy | 26 |
| 4.6 | Acierto frente al parámetro C en Dynamic Thompson Sampling | 27 |
| 4.7 | Tiempo de ejecución frente al parámetro k en Dynamic Thompson Sampling | 28 |
| 4.8 | Acierto y novedad para distintos valores de k en Dynamic Thompson Sampling | 29 |
| 4.9 | Acierto comparativa de algoritmos | 35 |
| 4.10 | Novedad comparativa de algoritmos | 36 |

Lista de tablas

| | | |
|-----|--|----|
| 3.1 | Tabla de métricas | 22 |
| 4.1 | Tabla de comparativa de los algoritmos basados en ϵ -greedy | 30 |
| 4.2 | Tabla de comparativa de los algoritmos basados en UCB | 31 |
| 4.3 | Tabla de comparativa de los algoritmos basados en Thompson Sampling | 31 |
| 4.4 | Tabla de comparativa de otros algoritmos | 32 |
| 4.5 | Tabla comparativa final de algoritmos | 34 |

INTRODUCCIÓN

En esta sección se explica la motivación del trabajo 1.1, los objetivos generales y concretos que se buscan cumplir 1.2 y la estructura que sigue el documento 1.3.

1.1. Motivación

Siendo los sistemas de recomendación una tecnología relativamente joven, aún hay cabida para el desarrollo e investigación de nuevas técnicas y aproximaciones para resolver problema. Recientemente se han empezado a adaptar técnicas de los llamados **bandidos multi-brazo** que combinan explotación (del conocimiento actual sobre el usuario) y exploración (de más gustos del usuario) [Sutton and Barto, 2018]. Además de contentar al usuario recomendándole ítems de su interés, cada recomendación es una oportunidad para el sistema de obtener información que le ayude a mejorar las futuras recomendaciones. Este tipo de estrategias asumen un planteamiento en el que se comprende la dualidad de objetivo de la recomendación: la explotación de las preferencias conocidas del usuario para complacerle, y la exploración de nuevos gustos para mejorar el acierto y ampliar el valor aportado al usuario a largo plazo. Este planteamiento intuitivamente se adapta mucho mejor al problema de recomendación que los sistemas de recomendación que se usan en el presente.

En este trabajo el análisis que se va a hacer se centra en los bandidos multi-brazo con exploración dinámica. Los bandidos multi-brazo con exploración dinámica la regulan de manera que deciden, a lo largo de la ejecución, cuando es buen momento para explorar más y cuando menos (a diferencia de la exploración estática que siempre mantiene un grado de exploración fijo). Esto permite al algoritmo sacrificar rendimiento en determinados momentos con el objetivo de poder aprovechar el conocimiento adquirido en el futuro. Se ha mostrado que la exploración dinámica puede tener resultados positivos [Vermorel and Mohri, 2005a], aun así, es un aspecto de los algoritmos de bandidos multi-brazo poco explorado [Caelen and Bontempi, 2008]. Para poder medir el rendimiento se va a hacer uso de métricas de novedad y diversidad además de tiempo de ejecución y acierto.

1.2. Objetivos

En este trabajo se aborda la construcción de unos experimentos utilizando bandidos multi-brazo con exploración dinámica con el objetivo de evaluar su rendimiento y compararlo con los resultados que se obtienen al utilizar exploración estática. Para ello se plantean los siguientes objetivos:

- Estudio y comprensión teórica de la exploración dinámica y las aproximaciones existentes en esta rama.
- Implementación de los algoritmos contextuales (LinUCB y CLUB) y adversarios (EXP3) para utilizar como *baseline* además de algoritmos de exploración estática implementados por el estudiante en trabajos previos (ϵ -greedy, Upper Confidence Bound (UCB) y Thompson Sampling).
- Implementación de los algoritmos Adaptive ϵ -greedy, VDBE ϵ -greedy, ϵ -first greedy, Dynamic Thompson Sampling y AdaptiveUCB (creación propia) para la realización de los experimentos.
- Análisis de sensibilidad de los hiperparámetros de los algoritmos de exploración dinámica implementados.
- Evaluación del rendimiento y comparación de los distintos algoritmos y sus conclusiones.

1.3. Estructura del documento

La memoria está organizada de la siguiente manera:

Capítulo 1: Introducción. En este capítulo se describe cuál es la motivación que conduce a la realización de este trabajo y los objetivos que se persiguen conseguir.

Capítulo 2: Estado del arte. En este capítulo se explican las bases de los sistemas de recomendación actuales, el concepto de aprendizaje por refuerzo en general y su aplicación en los sistemas de recomendación en concreto, haciendo hincapié en la exploración dinámica.

Capítulo 3: Diseño y desarrollo. Aquí se muestra el procedimiento de implementación que se ha utilizado, se explican los algoritmos que se utilizan y las métricas que miden el rendimiento.

Capítulo 4: Experimentos y resultados. Esta sección se divide en 2 experimentos distintos. Primero una búsqueda y análisis de sensibilidad de los hiperparámetros para cada uno de los algoritmos que hacen uso de exploración dinámica y más adelante el experimento que compara los resultados de estos algoritmos respecto a los que utilizo como *baseline* (exploración estática y filtrado colaborativo).

Capítulo 5: Conclusiones y trabajo futuro. En este último capítulo se expone de forma sin-

tetizada la conclusión del desarrollo y los experimentos realizados en este trabajo y se plantean posibles direcciones de trabajo futuro.

ESTADO DEL ARTE

En este capítulo se explican los sistemas de recomendación actuales 2.1, el aprendizaje por refuerzo en general 2.2 junto a los bandidos multi-brazo en concreto 2.2.1 (haciendo especial énfasis en la exploración dinámica 2.2.2).

2.1. Sistemas de recomendación

Los sistemas de recomendación son las técnicas y herramientas *software* que proporcionan sugerencias de ítems a usuarios [Burke, 2007]. Estas sugerencias se hacen a través de la estimación del *rating* esperado que le va a dar el usuario a todos los ítems con los que aún no ha interactuado. Su objetivo principal es mejorar la experiencia de los usuarios en el sistema [Ricci et al., 2010].

Estos últimos años la recomendación se ha vuelto más relevante. Se ha dado un crecimiento enorme en la web y por tanto pasa a ser más útil conocer que información le interesa a un usuario y cuál no. Es muy común encontrar sistemas de recomendación en los principales servicios de la web como e compra de productos en *Amazon* o *eBay*, recomendación de contenido en redes sociales como *Facebook* o *Twitter* o generación de listas de reproducción de música y vídeo en servicios como *Netflix*, *Youtube* o *Spotify* [Ricci et al., 2010].

2.1.1. Tarea de recomendación

Un sistema de recomendación tiene que:

- **Observar** a los usuarios comportarse en el sistema mientras evalúan ítems.
- **Detectar** patrones de comportamiento e identificar los gustos e intereses de cada usuario.
- **Predecir y recomendar** al usuario ítems que considera que le pueden interesar basado en el análisis anterior.

Lo que el sistema debe hacer es generar un *ranking* de todos los ítems con los que el usuario aún no ha interactuado. Este *ranking* se genera mediante algoritmos que estiman el valor esperado de cada

uno de los ítems y se ordenan de mayor a menor. Normalmente se recomienda más de un ítem y por tanto se eligen los top k elementos del *ranking* generado, siendo k el número de ítems que se quieren recomendar.

Los datos que necesita el sistema para resolver la tarea son las valoraciones pasadas de los usuarios a los ítems. Por ejemplo, si la aplicación permite puntuar los ítems de 1 a 10 siendo 1 lo peor y 10 lo mejor, hay que guardar las puntuaciones que ha dado cada usuario a cada ítem y trabajar con la tabla de datos resultante. En el caso de sistemas de compras de productos esto puede determinar que el usuario termine comprando el ítem o no y en el caso de contenido multimedia que el usuario termine accediendo al contenido o no. Hay muchas maneras de evaluar el interés del usuario en los ítems dependiendo del contexto en el que se esté utilizando el sistema. Es por ello que, en determinados contextos, ser capaces de evaluar el rendimiento del sistema de recomendación puede llegar a ser una tarea ardua difícil.

La figura 2.1 muestra un esquema de una posible base de datos de *ratings*. Se trata de una matriz de 2 dimensiones en la que se guardan las valoraciones (celdas) de los usuarios (filas) para cada ítem (columnas).













| | | <i>i</i> | | | | | | |
|----------|---|---|---|---|---|---|--|---|
| | | Items | | | | | | |
| | |  |  |  |  |  |  |  |
| <i>u</i> |  | 4 | | 4 | 2 | | 2 | 2 |
| |  | 1 | 4 | 4 | | 4 | | |
| |  | 4 | 3 | ? | 2 | 5 | ? | 2 |
| |  | 4 | 3 | 3 | | | 2 | 2 |
| |  | | 1 | 1 | 5 | 1 | 5 | 5 |

Figura 2.1: Esquema de la estructura de datos que maneja un sistema de recomendación. Se dispone de una base de datos donde se almacenan los *ratings* que los usuarios han dado a los diferentes ítems. Los ítems que aún no se le han mostrado al usuario son candidatos para recomendar y se debe calcular su puntuación esperada para así generar el *ranking*.

2.1.2. Objetivos de la recomendación

Ya hemos definido lo que es un sistema de recomendación, ahora queremos ilustrar con una serie de ejemplos los posibles objetivos que puede querer cumplir. Lo primero es diferenciar entre la utilidad que tiene el sistema para el que proporciona la recomendación de la utilidad que le proporciona al usuario [Ricci et al., 2010]. Por ejemplo, una agencia de viajes introduce un sistema de recomendación para así aumentar el número de ventas en habitaciones de hoteles [Borràs et al., 2014]. Mientras

tanto, el usuario se beneficia de acceder a los hoteles que mejor se adapten a sus necesidades. Hay innumerables razones por las que un proveedor de servicios puede querer utilizar un sistema de recomendación, pero las razones generales son [Ricci et al., 2010]:

- **Aumentar el número total de objetos vendidos:** Al adaptarse las recomendaciones mejor al perfil del usuario, este es más susceptible de realizar la compra. Este suele ser el objetivo general de los sistemas de recomendación comerciales. Este objetivo se puede generalizar a cualquier sistema que genere beneficio directa o indirectamente sin ser necesariamente ventas directas de productos. Otros ejemplos pueden ser *Youtube* recomendando vídeos que harán que aumentes el tiempo de visualización o una página web de noticias sugiriendo noticias a las que hacer clic. Mientras no se trate de un engaño o una manipulación del sistema hacia el usuario, este objetivo beneficia también a este ya que hará uso de un ítem que quiere y decide gastar tiempo o dinero en él.
- **Vender objetos más diversos:** Cuando se dispone de un ítem poco popular, el proveedor de servicios no puede enseñárselo a cualquiera y arriesgarse a perder ventas, sin embargo, puede recomendárselo a los usuarios más aptos a aceptar ese ítem (de nuevo, estos ítems pueden referirse a productos en venta, videos de *Youtube*, hoteles en una compañía de viajes o cualquier servicio que al proveedor le interese promocionar). De la misma manera que el punto anterior, esto beneficia al usuario ya que le permite descubrir ítems que de otra manera no habría encontrado.
- **Aumentar la satisfacción del usuario:** Cuando a un usuario se le recomiendan los ítems que le interesan y relevantes a la vez que se omiten los que no quiere ver, su experiencia en el sistema mejora. Un usuario satisfecho se traduce en una utilización del sistema más frecuente y por tanto mejores recomendaciones (que a su vez genera mayor satisfacción), mayores "ventas" en el sistema y más probabilidades de nuevos usuarios utilizándolo.
- **Aumentar la fidelidad del usuario:** Un usuario es más fiel a un sistema que le reconoce como un usuario de valor y le proporciona una vista personalizada. Cuanto más antiguo sea el usuario en el sistema, más información se dispone de él y por tanto mejor es la personalización y más fiel se vuelve.
- **Mejorar el entendimiento de lo que el usuario quiere:** El proveedor del servicio dispondrá de información valiosa sobre los gustos y preferencias de sus usuarios. Estos datos pueden utilizarse para tomar decisiones de negocio o campañas de *marketing*.

2.1.3. Técnicas de recomendación

Existen dos claras maneras de afrontar la recomendación. La **recomendación basada en contenido** y el **filtrado colaborativo**. La primera hace uso de las características de los ítems y de las preferencias de los usuarios sobre esas características mientras que la segunda se basa en la similitud

en las valoraciones de los usuarios a los ítems. Cada uno de ellos tiene sus ventajas e inconvenientes. También existe la recomendación híbrida donde se combinan distintas estrategias de recomendación (basado en contenido + filtrado colaborativo) pretendiendo resolver los problemas de cada uno de ellos.

Recomendación basada en contenido

El principio general de la **recomendación basada en contenido** es identificar las características comunes de los ítems que ya han recibido una valoración positiva por el usuario al que se le quiere recomendar [Balabanovic and Shoham, 1997]. El sistema necesita clasificar los ítems en distintas categorías dependiendo de sus características [Figueroa et al., 2004]. Si por ejemplo se trata de una aplicación de películas, se puede tener una base de datos de director, actores, géneros, etc. Este tipo de recomendación es capaz de recomendar ítems que sean nuevos o poco conocidos ya que no necesita que hayan sido puntuados previamente. Sin embargo, tiene el problema de que recomienda siempre cosas parecidas a las que el usuario ha valorado y por tanto se pierde el efecto de exploración. Un sistema basado solamente en contenido sufre problemas de análisis de contenido limitado y sobre especialización [Shardanand and Maes, 1995]. El problema del análisis de contenido limitado surge cuando el sistema posee información limitada del usuario o del contenido de los ítems, por ejemplo, el usuario decide no proporcionar información privada al sistema y por tanto la información es insuficiente para determinar el perfil del usuario. También sacar las características de ítems como música o imágenes puede ser costoso y por tanto de nuevo, la información no es suficiente para determinar la calidad del ítem. La sobre especialización causa la recomendación de ítems demasiado parecidos en contenido a los que el usuario ha valorado positivamente [Ning et al., 2015].

En estos sistemas cada ítem se representa como un vector en el espacio de características y mediante una función de similitud, podemos medir cuán parecidos son dos ítems. En este trabajo en concreto para la similitud se utiliza el índice de Jaccard [Real and Vargas, 1996], pero existen otras como la similitud coseno.

Filtrado colaborativo

En vez de depender de información sobre el contenido de los ítems y sus características, el **filtrado colaborativo** se basa en la observación del resto de usuarios y sus gustos para recomendar ítems. Consiste en fijarse en el resto de los usuarios parecidos al usuario al que se le quiere hacer la recomendación y predecir su valoración de un ítem en función de la puntuación que le han dado los usuarios similares [Konstan et al., 1997]. El filtrado colaborativo supera algunas de los problemas que tiene la recomendación basada en contenido. Por ejemplo, ítems de los cuales es difícil sacar características ya no son un problema porque no hacen falta [Ning et al., 2015].

El filtrado colaborativo se puede aproximar de dos maneras. Los métodos basados en vecinos

(knn), que almacena las valoraciones de los usuarios a los ítems y las utiliza para predecir *ratings* de nuevos ítems, y los métodos basados en modelo que son muchos y muy variados: regresión, redes neuronales, factorización de matrices, entre otros. Esta aproximación en general da buenos resultados, pero presenta problemas cuando aún no se conocen datos de usuarios y de ítems. Se necesita que los usuarios hayan puntuado unos cuantos ítems y que los ítems hayan sido puntuados para que los resultados sean fiables. Si no, puede ocurrir que los usuarios no tengan suficientes vecinos o que estos no hayan visto nada nuevo que ellos no conozcan.

Knn en filtrado colaborativo tiene dos aproximaciones distintas pero muy parecidas. La primera es knn colaborativo basado en ítem. Sigue la misma filosofía y estructura que el knn explicado en la recomendación basada en contenido salvo porque la similitud no se basa en las características de los ítems sino en los *ratings* que los usuarios le han dado. La segunda aproximación es knn colaborativo basado en usuario [Goldberg et al., 1992]. Esta aproximación es la que se utiliza en los experimentos como *baseline* de un algoritmo enteramente codicioso. Consiste en coger los k usuarios más parecidos al usuario al que se le quiere hacer la recomendación y utilizar los *ratings* de los ítems que han puntuado esos usuarios similares y el usuario no [Ekstrand et al., 2011]. En la figura 2.2 se muestra un esquema del funcionamiento.

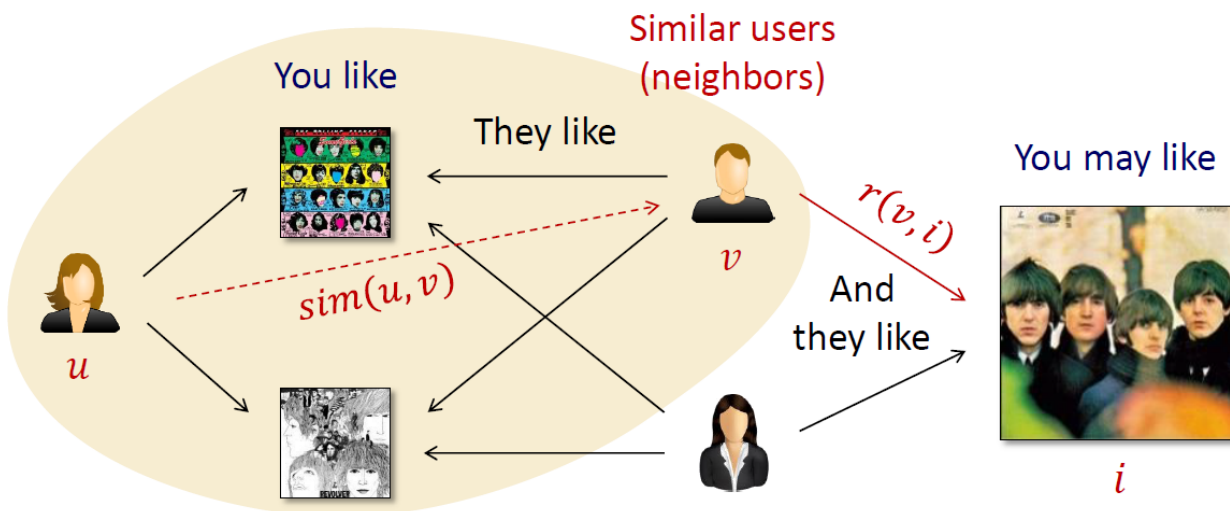


Figura 2.2: Esquema del funcionamiento de filtrado colaborativo knn con similitud entre usuarios. En este caso se debe calcular la similitud de todos los usuarios con el usuario al que queremos realizar la recomendación en este momento. Se escogerán los k usuarios más similares y se aplicará una función para calcular el *rating* esperado de cada uno de los ítems no evaluados por el usuario, pero sí por al menos uno de sus vecinos. Finalmente, se le recomendará el ítem con el mayor *rating* esperado.

2.2. Aprendizaje por refuerzo

El aprendizaje por refuerzo es aprender a que hacer (asociar situaciones a acciones concretas) con el fin de maximizar una recompensa numérica. Más concretamente, es un enfoque computacional al entendimiento del aprendizaje con objetivo y la toma de decisiones [Sutton and Barto, 2018]. Se inspira en el aprendizaje que desarrollan los seres vivos. Un ser vivo que no conoce su entorno comienza tomando decisiones aleatorias y aprendiendo poco a poco mediante prueba y error a tomar las que le generan recompensas positivas, aunque no deje de explorar opciones nuevas. Cuando un ser vivo realiza acciones en un entorno, se crea una conexión sensomotora con el entorno que produce una riqueza de información sobre causa y efecto ideal para la toma de decisiones [Bertsekas and Tsitsiklis, 1995]. El problema se puede construir genéricamente como un **agente** que toma determinadas **acciones** en un **entorno** y obtiene una **recompensa** que utiliza para aprender. Se diferencia de otras herramientas de aprendizaje automático en su énfasis en dejar al agente interactuar con el entorno sin supervisión ni información completa del entorno [Sutton and Barto, 2018].

2.2.1. Formulación del problema

Procesos de decisión de Markov finitos

Los procesos de decisión de Markov, MDP (*Markov Decision Process*) a partir de ahora, siguen una formulación igual a la general salvo porque se encuentran en un **estado** determinado. En la figura 2.3 se muestra un esquema del planteamiento. Al realizar una acción, el agente puede cambiar de estado y por tanto se pasa a tener acceso a distintas recompensas. Por tanto, los MDP implican una recompensa retrasada a cambio de una recompensa inmediata [Sutton and Barto, 2018].

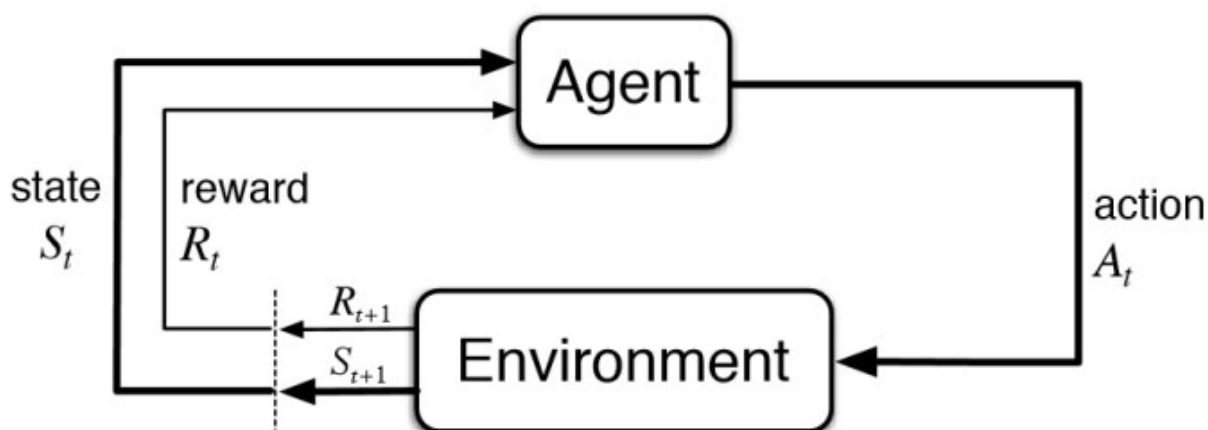


Figura 2.3: Esquema de un proceso de decisión de Markov. El agente realiza determinadas acciones que generan una recompensa y modifican el estado en el que se encuentra el entorno [Sutton and Barto, 2018].

"Más concretamente, el agente y el entorno interactúan durante una secuencia de instantes en el tiempo $t = 0, 1, 2, \dots$. En cada uno de estos momentos el agente recibe una representación del estado en el que se encuentra el entorno S_t del total de estados S . Con esta información el agente elige una acción A_t del conjunto de acciones posibles en ese estado $A(s)$. En el siguiente momento de tiempo, como respuesta de su acción el agente recibe una recompensa R_{t+1} como un valor numérico real. El entorno cambia a un nuevo estado S_{t+1} y se repite el proceso. Por tanto, el entorno y el agente generan una secuencia que comienza de esta manera: $S_0, A_0, R_1, S_1, A_1, R_2, \dots$. Al ser finito, el número de estados, acciones y recompensas también es finito." [Aróstegui and Castells, 2020]

Bandidos multibrazo

Como simplificación de los MDP existen los bandidos multi-brazo que no incluyen distintos estados. En este caso un bandido (agente) se enfrenta a una serie de palancas que accionan máquinas tragaperras (acciones) y deberá decidir que palanca le interesa accionar en cada iteración, recibiendo una recompensa monetaria en cada una de las máquinas [Ek and Strigsson, 2015]. El bandido deberá compensar el escoger las acciones que le proporcionan la máxima recompensa (explotación) y por tanto maximizar la recompensa a corto plazo o bien escoger otras palancas (exploración) para así optar a recompensas mayores y maximizar la recompensa total a largo plazo.

Cada una de las posibles palancas que el agente puede accionar tiene una recompensa esperada llamada *valor*. Si conociéramos los valores exactos de cada acción, el problema se resuelve de manera trivial, solamente hay que elegir la acción con el valor máximo. Sin embargo, no se conocen estos valores y se deben ir ajustando. Para estimar estos valores se puede hacer la media de recompensas de cada acción:

$$Q_t(a) = \frac{\text{suma de recompensas de } a \text{ antes de } t}{\text{numero de veces que se ha realizado } a \text{ antes de } t}$$

Con estos datos se plantean diferentes estrategias para escoger las acciones según estos valores esperados. Existen muchos y diferentes algoritmos que afrontan este problema de diferentes maneras. En este trabajo se han utilizado como *baseline* de algoritmos de bandidos multi-brazo con exploración estática (implementados en trabajos anteriores [Aróstegui and Castells, 2020]) ϵ -greedy [Sutton and Barto, 2018], Upper Confidence Bound [Auer et al., 2002a], Thompson Sampling [Chapelle and Li, 2011]. Como *baseline* de algoritmos de bandidos multi-brazo adversarios se ha implementado EXP3 [Auer et al., 2002b] y como *baseline* de bandidos multi-brazo contextuales se han implementado LinUCB [Li et al., 2010] [Mehrotra et al., 2020] y CLUB [Gentile et al., 2014].

2.2.2. Exploración dinámica en bandidos multibrazo

El principal dilema al que se enfrentan los algoritmos de bandidos multi-brazo es el balanceo entre exploración y explotación. Explotar te permite obtener la recompensa más alta en este preciso instante,

pero demasiada previene la maximización de la recompensa a largo plazo. Explorar escoge un brazo aleatorio independientemente de las recompensas pasadas consiguiendo de esta manera optar por recompensas mayores [Hao et al., 2020] aunque demasiada exploración no permite aprovechar las mejores recompensas observadas hasta el momento. Los algoritmos de bandidos multi-brazo como ϵ -greedy tienen un valor de exploración fijo, es decir, explora con la misma probabilidad a lo largo de toda la ejecución. Algoritmos como Thompson Sampling o Upper Confidence Bound tienen una evolución dinámica del término exploratorio que les permite adaptarse al momento de la ejecución en el que se encuentren. No obstante, en la versión estática de estos algoritmos, el régimen de adaptación es siempre el mismo y no es configurable. Una modificación permitiría añadir dinamismo a este régimen.

Existen distintas maneras de añadir dinamismo a la exploración en un bandido multi-brazo. Si tomamos ϵ -greedy como ejemplo podemos modificar el valor ϵ de distintas maneras. Se puede ir disminuyendo el valor de la exploración a lo largo del tiempo mediante una función como se verá en la sección 3.3.2. Elegir el valor de ϵ dinámicamente a lo largo de la ejecución según la información que tenemos sobre el estado actual como en la sección 3.3.1. También se puede tener una primera fase de exploración pura para luego tener una segunda de explotación pura (o una exploración menos agresiva) como se explicará 3.3.3.

Además de añadir dinamismo a algoritmos que no lo tienen, también cabe estudiar modificaciones en la forma de modificar los parámetros en algoritmos que ya de por sí son dinámicos pero que pueden beneficiarse de ciertas modificaciones. En las secciones 3.3.4 y 3.3.5 se describirán modificaciones de Thompson Sampling y Upper Confidence Bound respectivamente donde se modifican el dinamismo de su exploración.

Se ha probado que bajo condiciones específicas la exploración dinámica rinde mejor que las variantes estáticas [Vermorel and Mohri, 2005a] pero es un aspecto de los bandidos multi-brazo poco profundizada [Caelen and Bontempi, 2008].

DISEÑO Y DESARROLLO

En este capítulo se explica toda la implementación que se ha realizado del escenario de pruebas 3.1, se hace un análisis de los datos utilizados para los experimentos 3.2, se explican en profundidad los algoritmos de bandidos multibrazo con exploración dinámica 3.3 y por último se explican las métricas que se van a utilizar para su evaluación 3.4.

3.1. Implementación y planteamiento

Basándose este trabajo en uno anterior, el escenario de pruebas genérico ya estaba creado y solamente ha sido necesaria la implementación de los nuevos algoritmos que se quieren evaluar. Como una de las métricas que se va a medir es la del tiempo de ejecución, se ha hecho especial énfasis en la eficiencia del código en cada uno de los algoritmos, por ejemplo, haciendo uso de programación funcional. Como referencia para cada uno de los algoritmos se han utilizado los artículos correspondientes a cada uno de ellos y para enfoques más generales he utilizado [Castells and Sanz-Cruzado, 2019] y [Galbraith, 2016].

El escenario de evaluación de los algoritmos consiste en un proceso continuo en el que se recorren todos los usuarios de la base de datos de manera aleatoria y se les va recomendando un ítem cada vez, iterando sobre todos los usuarios 100 veces. Dependiendo de la respuesta (consulta a la base de datos) del usuario se considerará si el sistema ha acertado con la recomendación o no. Es decir, la base de datos solamente se utiliza para evaluar los algoritmos en el momento de ejecución.

3.2. Análisis de los datos

La base de datos que se utiliza en los experimentos es MovieLens 1M [Movielens, 1997]. Consiste en una base de datos recogida por GroupLens Research [GroupLens, 1997] en el año 2003. Mas concretamente, se trata de una colección de 1.000.209 ratings procedentes de 6040 usuarios en 3900 películas. Las valoraciones de los usuarios son un número entre 1 y 5, siendo 1 la peor puntuación y 5 la mejor. El *threshold* utilizado en este trabajo para separar si a un usuario le ha gustado o no una

película ha sido la puntuación 3. Además, se proporciona un fichero de etiquetas para las películas que las categorizan en 14 géneros ya predefinidos. Gracias a estas etiquetas se pueden medir métricas de diversidad e implementar algoritmos como CLUB o LinUCB.

3.3. Bandidos multi-brazo con exploración dinámica

3.3.1. Adaptive ϵ -greedy

Adaptive ϵ -greedy [dos Santos Mignon and de Azevedo da Rocha, 2017] es un algoritmo que añade capacidad adaptativa a ϵ -greedy variando su valor según las recompensas pasadas. En el ϵ -greedy original se fija un valor ϵ que determina la probabilidad de explorar que tiene el bandido. Adaptive ϵ -greedy sigue la misma aproximación, pero en vez de tener un valor fijo para ϵ lo va modificando según avanza la ejecución haciéndolo crecer o disminuir según evolucionan las recompensas obtenidas. En la figura 3.1 se observa un ejemplo de evolución del valor de ϵ a lo largo de una ejecución.

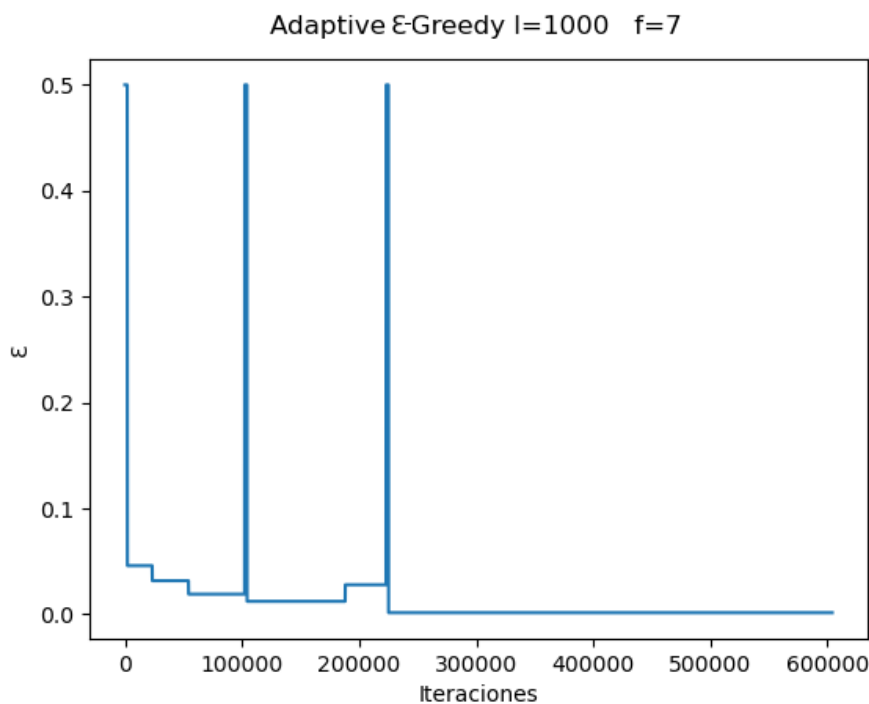


Figura 3.1: Ejemplo de posible evolución de ϵ a lo largo de una ejecución de Adaptive ϵ -greedy. En este caso se muestra una ejecución en el que la condición para modificar ϵ se comprueba cada 1000 iteraciones. Se observa un decrecimiento general con picos donde se reinicia el valor para continuar explorando. Una vez el algoritmo considera que ha explorado suficiente, ϵ deja de modificarse y se mantiene en un valor bajo.

El algoritmo dispone de dos hiperparámetros: l y f . El parámetro l se utiliza para determinar el número de acciones exploratorias que se deben tomar antes de evaluar si el valor de ϵ debe cambiar

o no. El parámetro f sirve para obtener valores adecuados al calcular el nuevo valor de ε [dos Santos Mignon and de Azevedo da Rocha, 2017]. En el algoritmo 3.2 se muestra el pseudocódigo del algoritmo. El valor de ε empieza siempre en 0.5 y cada l acciones exploratorias se evalúa si está rindiendo mejor que en las l anteriores o no. Si se cumple que está rindiendo mejor, se actualiza el valor de ε utilizando el parámetro de regularización f , si no, el valor de ε se mantiene fijo ya que significa que ha estado eligiendo ítems no óptimos y debe continuar explorando. En cualquier caso, se sigue manteniendo el funcionamiento normal de ε -greedy recomendando aleatoriamente si el valor aleatorio generado es menor o igual a ε y el ítem con el valor esperado más alto en caso contrario.

```

input :  $\varepsilon$ , usuario,  $l$ ,  $f$ 
output: Ítem recomendado para el usuario
1  maxprev  $\leftarrow$  0;
2  k  $\leftarrow$  0;
3  if random[0,1]  $\leq \varepsilon$  then
4    maxcurr  $\leftarrow$   $Q_t(A^*_t)$ ;
5    k  $\leftarrow$  k + 1;
6    if k =  $l$  then
7       $\Delta \leftarrow$  (maxcurr - maxprev) *  $f$ ;
8      if  $\Delta > 0$  then
9         $\varepsilon \leftarrow \text{sigmoid}(\Delta)$ 
10     end
11     else
12       if  $\Delta < 0$  then
13          $\varepsilon \leftarrow 0.5$ 
14       end
15       maxprev  $\leftarrow$  maxcurr;
16       k  $\leftarrow$  0;
17     end
18   end
19   return RandomChoice()
20 end
21 else
22   return ItemPuntuacionMaxima()
23 end

```

Algoritmo 3.1: Pseudocódigo del algoritmo *Adaptive ε -greedy* [dos Santos Mignon and de Azevedo da Rocha, 2017].

3.3.2. VDBE ε -greedy

VDBE ε -greedy [Tokic, 2010] es otra adaptación dinámica del algoritmo ε -greedy original [Sutton and Barto, 2018] que controla la probabilidad de exploración dependiendo de una función de error. El comportamiento deseado es que el agente explore más en situaciones donde el conocimiento es limitado, por ejemplo, al comienzo de la ejecución. Por otro lado, la exploración deberá disminuir según el conocimiento aumenta y se tiene mayor certeza del entorno. Este comportamiento se puede observar

en la figura 3.2 que muestra un ejemplo de la evolución de ϵ a lo largo de una ejecución.

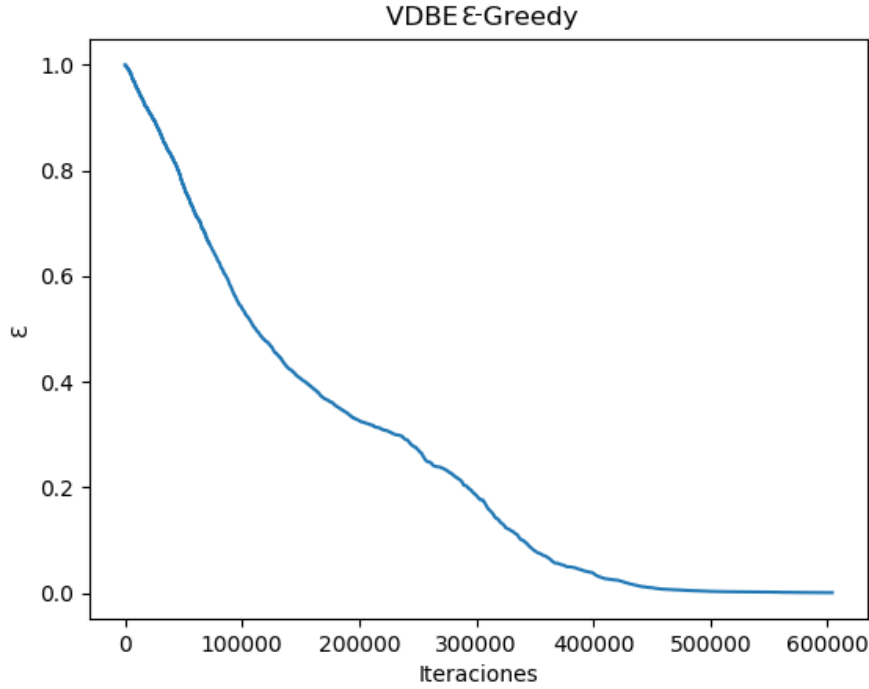


Figura 3.2: Ejemplo de posible evolución del valor de ϵ a lo largo de una ejecución completa de VDBE ϵ -greedy. Se observa un decrecimiento constante de la probabilidad de explorar a lo largo de la ejecución. Modificando los hiperparámetros δ y σ podremos acelerar esta caída o frenarla.

Para conseguir este comportamiento se hace uso de una distribución Boltzmann que se calcula en cada iteración según la ecuación 3.1:

$$f(s_t, a_t, \sigma) = \frac{1 - e^{-\frac{|Q_{t+1}(s,a) - Q_t(s,a)|}{\sigma}}}{1 + e^{-\frac{|Q_{t+1}(s,a) - Q_t(s,a)|}{\sigma}}} \quad (3.1)$$

Comenzando con $\epsilon_{t=0}(s) = 1$ la regla para actualizar el valor de ϵ cada iteración es 3.2:

$$\epsilon_{t+1}(s) = \delta \cdot f(s_t, a_t, \sigma) + (1 - \delta) \cdot \epsilon_t(s) \quad (3.2)$$

Donde σ es un hiperparámetro mayor que 0 llamado sensibilidad inversa. Si tiene un valor bajo causa exploración y si es grande la ratio de exploración converge a 0 y resulta en una estrategia totalmente codiciosa. δ es otro hiperparámetro que determina la influencia de la acción seleccionada en la ratio de exploración.

```

input :  $\varepsilon$ , usuario,  $\sigma$ ,  $\delta$ 
output: Ítem recomendado para el usuario
1   $\varepsilon \leftarrow \delta \cdot f(\sigma) + (1 - \delta) \cdot \varepsilon$ ;
2  if  $\text{random}[0,1] \leq \varepsilon$  then
3    | return RandomChoice()
4  end
5  else
6    | return ItemPuntuacionMaxima()
7  end

```

Algoritmo 3.2: Pseudocódigo del algoritmo VDBE ε -greedy [Tokic, 2010]

3.3.3. ε -first Greedy

ε -first greedy [Even-Dar et al., 2002] es un algoritmo que también se basa en el ε -greedy original [Sutton and Barto, 2018] que mantiene $\varepsilon = 1$ durante las primeras x iteraciones para luego asignarle un valor codicioso (0.1, 0.001 o incluso 0). De esta manera forzamos al agente a explorar en el comienzo para luego explotar el conocimiento adquirido en esas primeras ejecuciones. El hiperparámetro que se puede ajustar en este algoritmo es *iterations* que determina cuantas iteraciones se quiere explorar antes de explotar [Vermorel and Mohri, 2005b]. En este trabajo, al haber mostrado en trabajos anteriores que el valor de ε óptimo es 0.1, se ha fijado el valor de ε en las iteraciones de explotación a ese valor. En el algoritmo 3.3 se muestra el pseudocódigo.

```

input :  $\varepsilon$ , usuario, iterations, epoca
output: Ítem recomendado para el usuario
1  if  $epoca > iterations$  then
2    |  $\varepsilon \leftarrow 0.1$ 
3  end
4  else
5    |  $\varepsilon \leftarrow 1$ 
6  end
7  if  $\text{random}[0,1] \leq \varepsilon$  then
8    | return RandomChoice()
9  end
10 else
11 | return ItemPuntuacionMaxima()
12 end

```

Algoritmo 3.3: Pseudocódigo del algoritmo ε -first greedy [Vermorel and Mohri, 2005b].

3.3.4. Dynamic Thompson Sampling

Dynamic Thompson Sampling [Gupta et al., 2011] es una variación del algoritmo Thompson Sampling original [Chapelle and Li, 2011]. En Thompson Sampling se muestrea una distribución de proba-

bilidad para cada ítem utilizando la distribución Beta con parámetros $\alpha_e(i)$, $\beta_e(i)$ siendo $\alpha_e(i)$ el número de aciertos para ese ítem y $\beta_e(i)$ el número de fallos. La estrategia para elegir el ítem sigue la fórmula 3.3 [Ek and Strigsson, 2015].

$$i_e = \underset{i}{\operatorname{argmax}} [\operatorname{beta}(\alpha_e(i) + \alpha_0, \beta_e(i) + \beta_0)] \quad (3.3)$$

La variación que propone Dynamic Thompson Sampling es la de no permitir que las distribuciones de probabilidades se estrechen demasiado limitando el crecimiento de $\alpha_e(i)$ y $\beta_e(i)$ [Sun and Li, 2020]. En la figura 3.3 se observa el comportamiento de la distribución según avanzan las iteraciones.

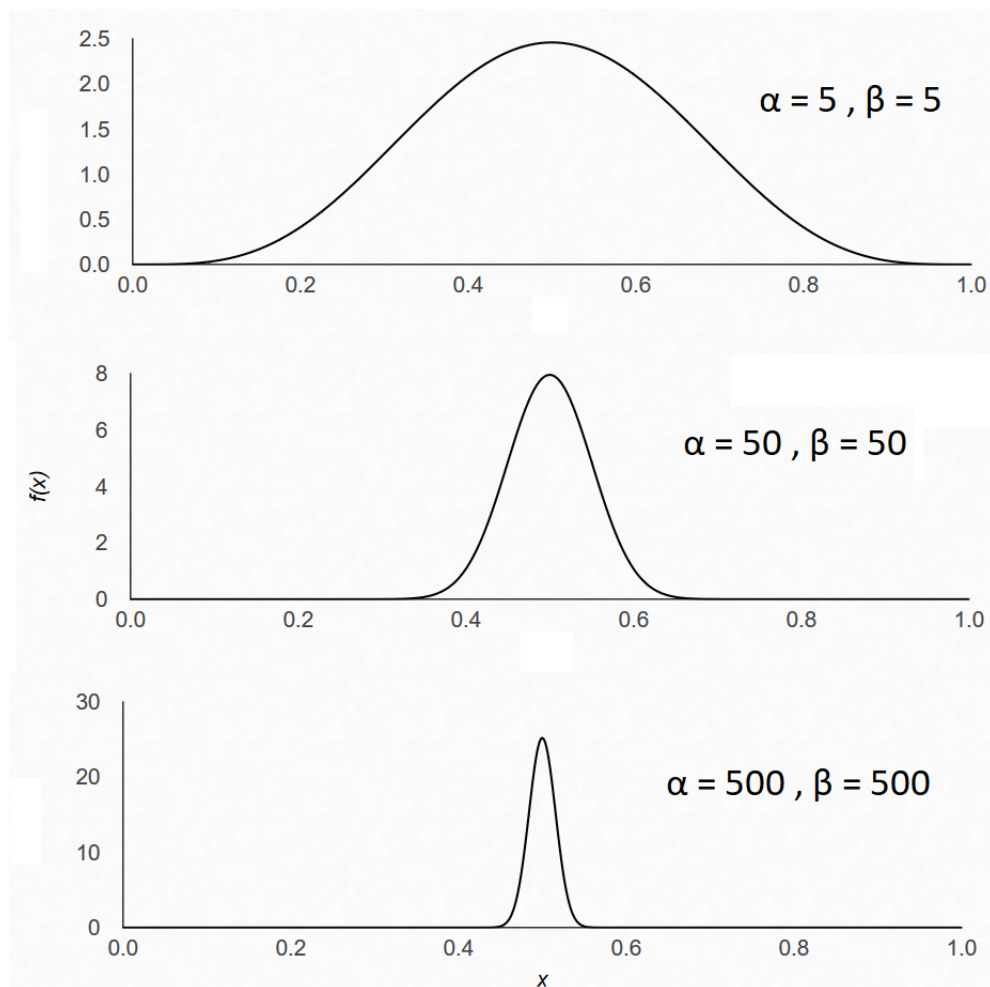


Figura 3.3: Evolución de la distribución Beta cuando los valores de $\alpha_e(i)$ y $\beta_e(i)$ aumentan. Se puede observar que la curva se estrecha, haciendo que sea poco probable escoger otros ítems que no sea el de la recompensa esperada más alta cuando $\alpha_e(i)$ y $\beta_e(i)$ son números grandes. La distribución puede moverse también en el eje x cuando $\alpha_e(i)$ y $\beta_e(i)$ son diferentes, pero aquí se pretende mostrar el efecto de estrechamiento.

Para evitar este estrechamiento lo que se hace es actualizar $\alpha_e(i)$ y $\beta_e(i)$ de tal manera que su suma no sobrepase un límite indicado por el hiperparámetro C . De esta manera le estamos dando más peso

a las recompensas más recientes y por tanto hacemos un seguimiento de las recompensas actuales. Para ello se realiza la comprobación cada iteración y en caso de sobrepasar el límite se actualizan de esta manera:

```

input :  $\alpha_e(i)$ ,  $\beta_e(i)$ , reward  $r$  y  $C$ 
output: Valores de  $\alpha_e(i)$  y  $\beta_e(i)$  actualizados
1 if  $\alpha_e(i) + \beta_e(i) < C$  then
2   |  $\alpha_e(i) \leftarrow \alpha_e(i) + r$ ;
3   |  $\beta_e(i) \leftarrow \beta_e(i) + 1 - r$ ;
4 end
5 else
6   |  $\alpha_e(i) \leftarrow (\alpha_e(i) + r) \frac{C}{C+1}$ ;
7   |  $\beta_e(i) \leftarrow (\beta_e(i) + 1 - r) \frac{C}{C+1}$ ;
8 end

```

Algoritmo 3.4: Pseudocódigo de la actualización de los parámetros de la distribución Beta en Dynamic Thompson Sampling [Sun and Li, 2020]. Este código se ejecuta para el ítem recomendado en cada iteración.

Originalmente la actualización de las distribuciones de probabilidad se realiza en cada iteración, pero en este trabajo se propone la utilización de un nuevo hiperparámetro k que sea capaz de regular el número de iteraciones que deben pasar antes de que se actualicen las distribuciones de nuevo. Con este hiperparámetro se pretende disminuir notablemente el tiempo de ejecución sin perder acierto. En la sección de análisis de hiperparámetros 4.1.4 se observa cómo afecta al comportamiento del algoritmo y se explica por qué en Dynamic Thompson Sampling funciona mejor que en Thompson Sampling.

```

input :  $\alpha_e(0)$ ,  $\beta_e(0)$ ,  $k$  y un usuario
output: ítem recomendado para el usuario
1  $iter \leftarrow iter + 1$ ;
2 if  $iter = k$  then
3   |  $iter \leftarrow 0$ ;
4   | foreach  $item\ i\ in\ dataset$  do  $X \leftarrow \text{BetaDist}(\alpha_e(i) + \alpha_e(0), \beta_e(i) + \beta_e(0))$ ;
5 end
6 return  $\max(X)$ 

```

Algoritmo 3.5: Pseudocódigo de Dynamic Thompson Sampling.

3.3.5. Adaptive UCB

AdaptiveUCB es un algoritmo desarrollado originalmente en este trabajo. Consiste en una modificación al algoritmo original de UCB [Auer et al., 2002a] la cual permite modificar de manera dinámica la importancia del término exploratorio (σ_e) de la fórmula 3.4 modificando el parámetro γ .

$$i_e = \underset{i}{\operatorname{argmax}} [s_e(i) + \gamma \sigma_e(i)] \quad (3.4)$$

El término exploratorio se calcula como:

$$\sigma_e = \sqrt{\frac{\ln(epoca)}{\alpha_e(i) + \beta_e(i)}} \quad (3.5)$$

En el algoritmo original el valor de γ es fijo y aunque el término exploratorio va disminuyendo de manera natural, este proceso se puede acelerar o frenar modificando el parámetro γ . Lo que propongo en este trabajo es calcular el valor de γ en todas las iteraciones como:

$$\gamma = \log\left(1 + \frac{\beta(e)}{\alpha(e)}\right) \quad (3.6)$$

Donde $\alpha(i)$ son el número de aciertos que llevamos y $\beta(i)$ el número de fallos que conocemos (de los ítems que hemos recomendado, los que tenían una valoración menor al *threshold*). De esta manera, al ser el número de aciertos mayor al número de fallos, el valor de gamma varía entre 0 y 1 (como ya se mostró en trabajos anteriores, un valor mayor a 1 da resultados no óptimos). Por lo tanto, el algoritmo no tiene ningún hiperparámetro que ajustar. En el algoritmo 3.6 se muestra el pseudocódigo del algoritmo y en la figura 3.4 se muestra un ejemplo de evolución del parámetro γ a lo largo de la ejecución.

```

input :  $\gamma$ , usuario
output: ítem recomendado para el usuario
1   $\gamma \leftarrow \log\left(1 + \frac{\text{fallos}}{\text{aciertos}}\right)$ ;
2  foreach ítem  $i$  in dataset do  $X \leftarrow \text{calculoFormula}(i)$ ;
3  return  $\max(X)$ 

```

Algoritmo 3.6: Pseudocódigo del algoritmo AdaptiveUCB.

3.4. Métricas de novedad y diversidad

Para medir el rendimiento de los algoritmos se van a hacer uso de algunas métricas de novedad y diversidad. La principal métrica que se quiere maximizar es el acierto, pero también se quiere poder ver otros comportamientos.

La novedad es la diferencia entre experiencias pasadas y presentes del usuario. Un sistema no-

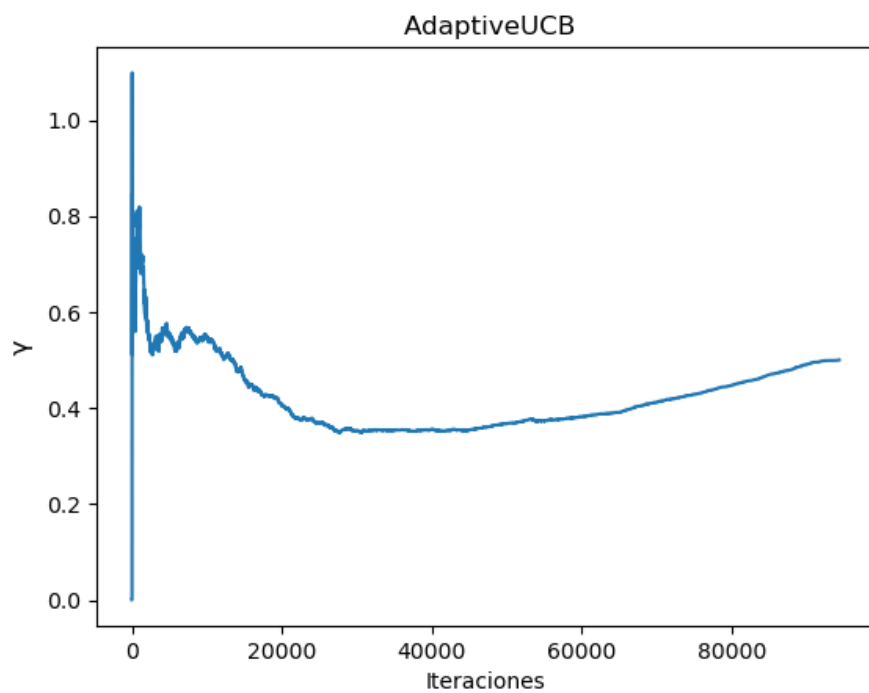


Figura 3.4: Ejemplo de posible evolución del valor de γ a lo largo de una ejecución completa de AdaptiveUCB. Se observa una variabilidad del valor grande al comienzo y como poco a poco converge a un valor aproximado de 0.5. Esto hace que la exploración al comienzo de la ejecución sea mayor pero que con el tiempo vaya disminuyendo.

vedoso tiende a mostrarnos elementos que no se nos hayan mostrado antes. La diversidad es la diferencia de las características internas de las partes de una misma experiencia. Un sistema diverso nos mostrará ítems diferentes entre ellos, por ejemplo, distintos géneros de películas. [Castells et al., 2015].

En la tabla 4.5 se muestran las fórmulas de las métricas que se van a utilizar siendo i y j ítems, R todos los ítems recomendados, I el conjunto total de ítems y $p(i_k)$ es la probabilidad de que se recomiende el k -ésimo ítem menos recomendado. Para las métricas *Intra-List Diversity* y *Unexpectedness* se utilizan los géneros de películas proporcionados en la base de datos y la similitud Jaccard [Real and Vargas, 1996] para calcular distancias. Además, se calculan como @10, es decir, observamos solamente los últimos 10 ítems recomendados.

| Métrica | Fórmula | Descripción |
|-----------------------------|--|--|
| Novedad | $\frac{1}{ R } \sum_{i \in R} (1 - \frac{ i^+ }{ i })$ | Media de impopularidad de ítems recomendados. |
| <i>Intra-List Diversity</i> | $\frac{1}{ R (R - 1)} \sum_{i \in R} \sum_{j \in R} d(i, j)$ | Media de las distancias de los ítems par a par. |
| <i>Unexpectedness</i> | $\frac{1}{ R I_u } \sum_{j \in R} \sum_{i \in I_u} d(i, j)$ | Media de las distancias del ítem con anteriores. |
| Coeficiente Gini | $\frac{1}{ I - 1} \sum_{k=1}^{ I } (2k - N - 1)p(i_k)$ | Medida de desigualdad entre n° recomendaciones. |

Tabla 3.1: Tabla de las métricas de novedad y diversidad que se van a utilizar para medir el rendimiento de los algoritmos [Castells et al., 2015]. Además de estas métricas se va a medir el acierto y el tiempo de ejecución.

EXPERIMENTOS Y RESULTADOS

En esta sección se realizan el análisis de hiperparámetros de los algoritmos de bandidos multibrazo con exploración dinámica explicados anteriormente 4.1 para luego realizar un experimento con estos algoritmos optimizados además de los demás algoritmos que se utilizan como *baseline*.

4.1. Análisis de hiperparámetros

Para seleccionar los hiperparámetros de cada algoritmo se va a representar cada uno de ellos frente al acierto para analizar si existe alguna tendencia. Después de representarlos nos quedamos con el valor del hiperparámetro que mayor acierto nos haya proporcionado. Los valores de los hiperparámetros se han escogido haciendo uso de la búsqueda en rejilla. Este análisis de los hiperparámetros se ha realizado sobre una partición aleatoria del 20 % de los datos de la base de datos y más adelante se utilizará el 80 % restante para la comparativa entre algoritmos. Cabe mencionar que la escala de algunos de los ejes para algunos hiperparámetros está en escala logarítmica. AdaptiveUCB no se encuentra en esta sección ya que no dispone de ningún hiperparámetro que optimizar.

4.1.1. Adaptive ϵ -greedy

Adaptive ϵ -greedy dispone de dos hiperparámetros: f y *limit*. El primero se utiliza para que el cálculo del nuevo ϵ sea adecuado mientras que *limit* nos indica el número de exploraciones que se deben realizar antes de evaluar si el valor debe cambiar o no.

En la figura 4.1 se observa el comportamiento del hiperparámetro f respecto al acierto habiendo fijado *limit*. Los puntos no parecen seguir ninguna tendencia fija. En el artículo original [dos Santos Mignon and de Azevedo da Rocha, 2017] se utiliza el valor de $f = 7$ que en este caso es el valor que mejor resultados ha obtenido. Se han probado valores menores y mayores, pero al no conseguir un acierto mayor, se ha optado por utilizar $f = 7$ en el test.

En la figura 4.2 se muestra la evolución del acierto para distintos valores del hiperparámetro *limit*. En este caso si se observa una tendencia ya que, si el valor es bajo, estamos evaluando demasiadas

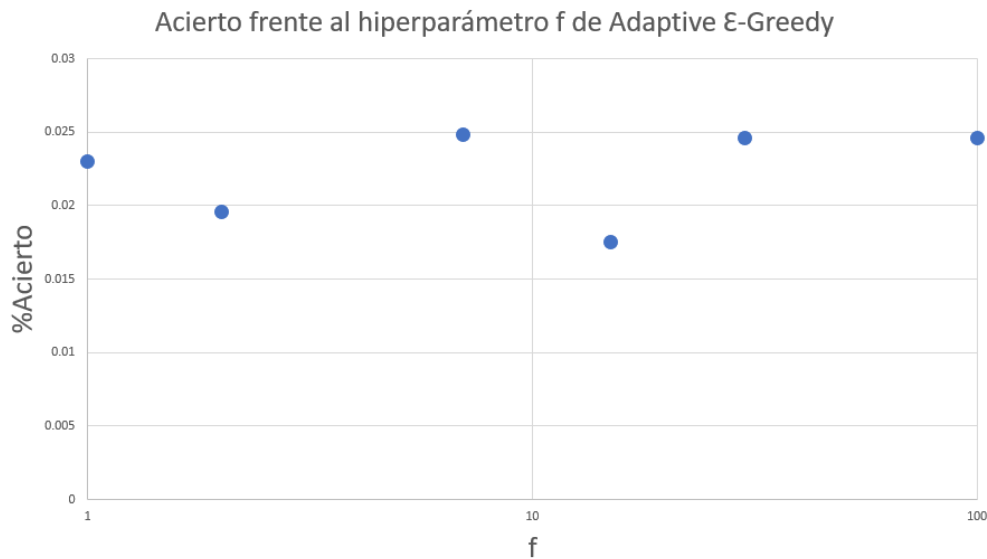


Figura 4.1: Gráfica de distintos valores de f frente al acierto en Adaptive ε -greedy. Eje x en escala logarítmica.

pocas iteraciones y por tanto el comportamiento es prácticamente aleatorio. Por otro lado, si el valor es demasiado alto, nos pasamos demasiadas iteraciones con $\epsilon = 0,5$ y por tanto no se llega a explotar el conocimiento. El valor con mejores resultados es $limit = 1000$ y es el que se va a utilizar en *test*.

4.1.2. VDBE ε -greedy

VDBE ε -greedy tiene dos hiperparámetros δ y σ . En la figura 4.3 observamos el comportamiento del acierto del algoritmo según modificamos el hiperparámetro δ . Podemos observar cómo hay un claro máximo en $\delta = 0,001$ que, tras sobrepasarlo, el acierto vuelve a caer a aciertos que se encontraban antes de ese valor. Salvo para valores extremos, parece que el algoritmo tiene un acierto base en 0.02 pero que $\delta = 0,001$ lo supera llegando a 0.025.

Por otro lado, en la figura 4.4 tenemos el comportamiento del acierto frente al valor de σ . El comportamiento es parecido al anterior, parece haber un máximo en $\sigma = 0,04$ que luego decae para valores más altos. El valor que se utilizará para test es $\sigma = 0,04$.

4.1.3. ε -first Greedy

ε -first greedy solamente dispone de un hiperparámetro al que he llamado *iterations*. Este hiperparámetro nos indica el número de iteraciones que el algoritmo va a explorar ($\epsilon = 1$) antes de comenzar a explotar ($\epsilon = 0,1$). En la gráfica 4.5 vemos el comportamiento del acierto frente a distintos valores del hiperparámetro. Podemos observar como el valor máximo se encuentra en $iterations = 10000$. Antes que eso, exploramos demasiado poco y cuando llegamos al momento de explotar, no disponemos de

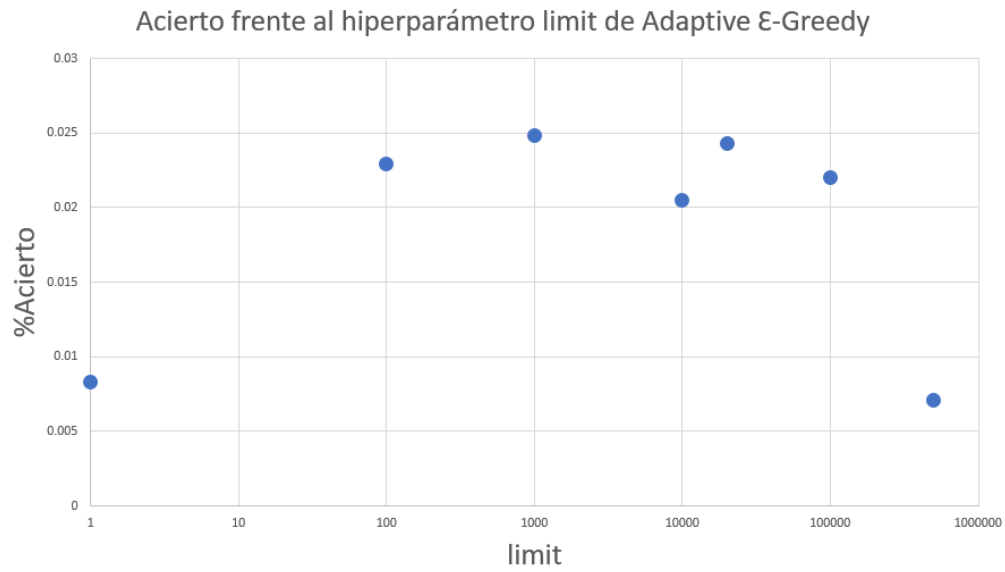


Figura 4.2: Gráfica de distintos valores de $limit$ frente al acierto en Adaptive ϵ -greedy. Eje x en escala logarítmica.

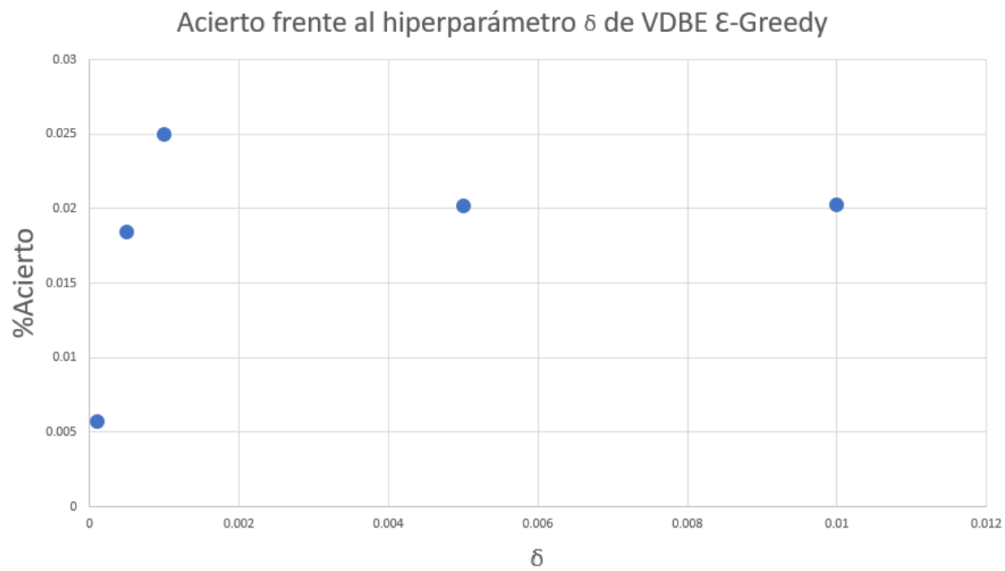


Figura 4.3: Gráfica de distintos valores del hiperparámetro δ frente al acierto en VDBE ϵ -greedy.

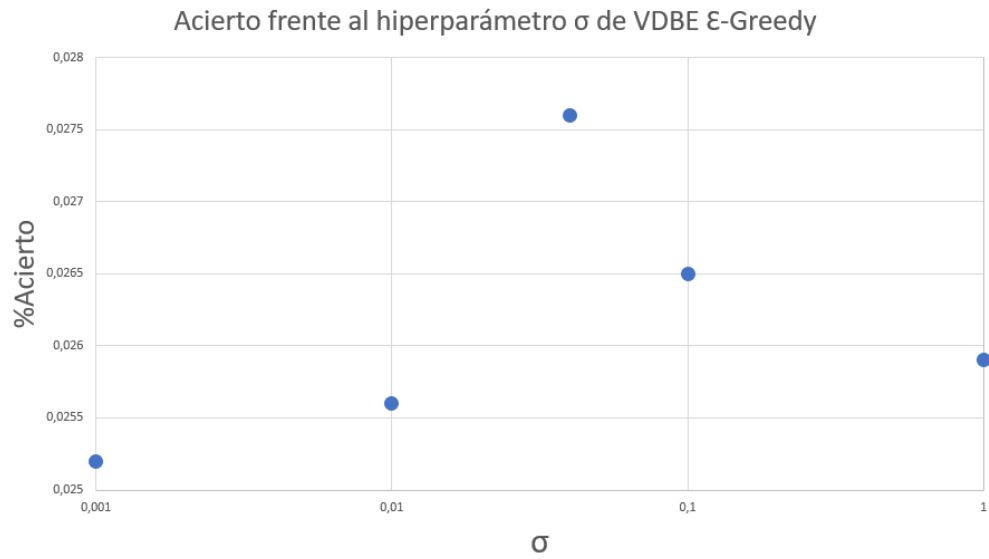


Figura 4.4: Gráfica de distintos valores del hiperparámetro σ frente al acierto en VDBE ϵ -greedy. Eje x en escala logarítmica.

suficiente información. Por el contrario, más que eso nos hace explorar demasiado y nos acercamos cada vez más a una recomendación totalmente aleatoria.

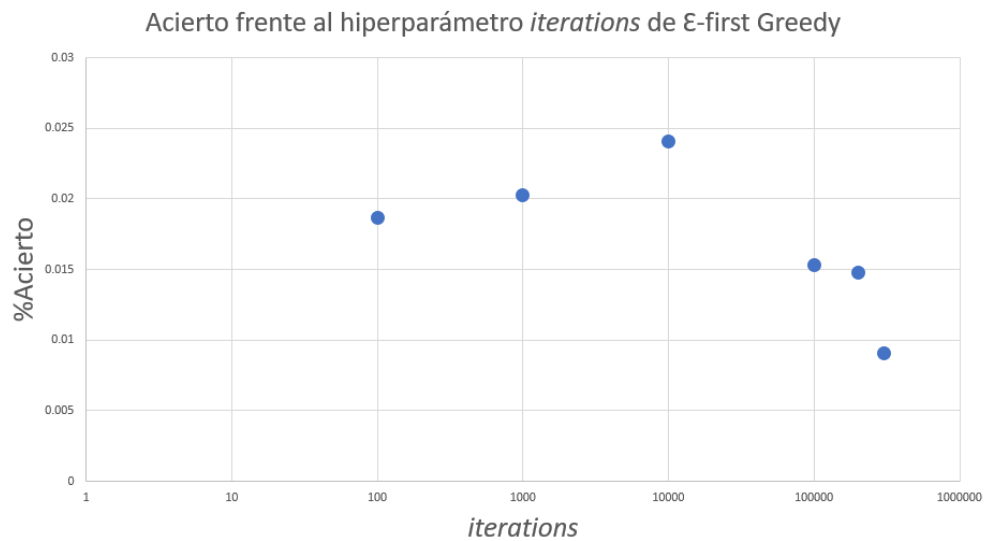


Figura 4.5: Gráfica de distintos valores de *iterations* frente al acierto en ϵ -first greedy. Eje x en escala logarítmica.

4.1.4. Dynamic Thompson Sampling

Dynamic Thompson Sampling dispone inicialmente de tres hiperparámetros: α_0 , β_0 y C . Más adelante se analiza cómo afecta el último hiperparámetro k del que se va a hacer un estudio aparte. De trabajos anteriores conocemos que para obtener buenos resultados α_0 tiene que ser bastante menor

que β_0 , es decir, una inicialización optimista. En este trabajo se ha decidido trabajar con los valores de $\alpha_0 = 1$, $\beta_0 = 20$ ya que no existe gran diferencia de acierto frente a valores como $\alpha_0 = 1$, $\beta_0 = 50$ y nos permite disminuir el valor de C más manteniendo la importancia de las primeras recompensas (si tenemos un C bajo y unos α_0 y β_0 que sumen más que ese valor, las primeras recompensas se verán muy mitigadas). Para observar con más detalle el rendimiento del algoritmo dirigirse a [Aróstegui and Castells, 2020].

Una vez tenemos fijado los valores de α_0 y β_0 es momento de analizar el comportamiento del nuevo hiperparámetro C . En la figura 4.6 podemos observar cómo evoluciona el acierto según modificamos el hiperparámetro. El valor máximo se encuentra en $C = 100$ aunque $C = 1000$ da valores muy próximos. Sin embargo, se observa claramente como según C aumenta, el acierto vuelve a decaer. Es de esperar que cuando C es suficientemente grande es como si no existiera el hiperparámetro (ya no hay un límite que no se permita sobrepasar) y el comportamiento del algoritmo es el de Thompson Sampling original. Es por esto por lo que se observa un valor prácticamente fijo a partir de 1000.

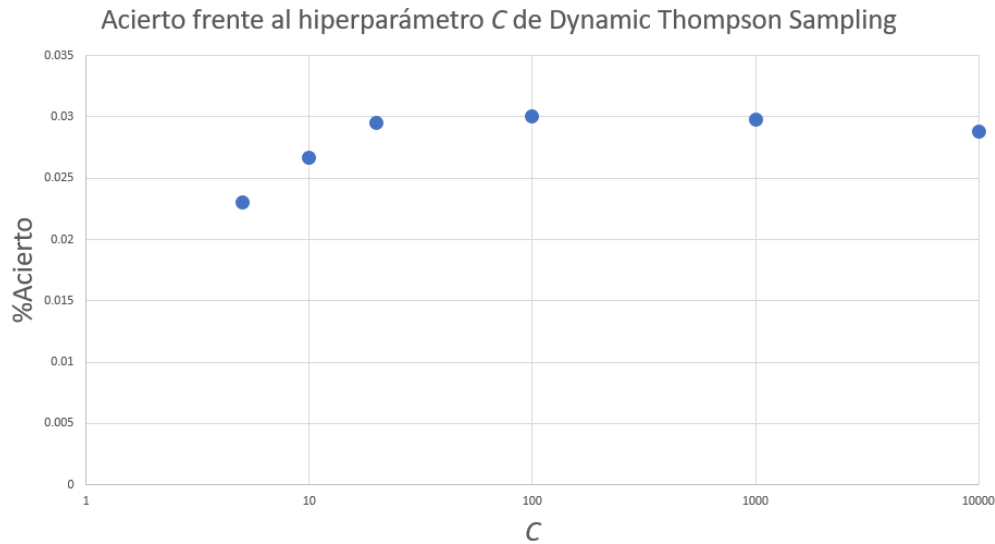


Figura 4.6: Gráfica de distintos valores del hiperparámetro C frente al acierto en Dynamic Thompson Sampling. Eje x en escala logarítmica.

El hiperparámetro k del algoritmo se presentó originalmente en [Aróstegui and Castells, 2020] para mejorar el tiempo de ejecución de Thompson Sampling sin perder acierto. Los resultados fueron satisfactorios ya que se consiguió aumentar ligeramente el acierto reduciendo a la mitad el tiempo de ejecución. Con esta nueva aproximación se ha probado y los resultados son todavía más positivos. En la figura 4.7 tenemos el tiempo que tarda el algoritmo en ejecutarse según distintos valores de k . La forma que tiene la curva es bastante intuitiva, el algoritmo tarda aproximadamente k veces menos en ejecutarse ya que tiene que hacer k veces menos iteraciones.

En [Aróstegui and Castells, 2020] se consiguió una mejora añadiendo este hiperparámetro tanto en tiempo de ejecución como en acierto para $k = 2$. En Thompson Sampling, cuando los valores de $\alpha_e(i)$

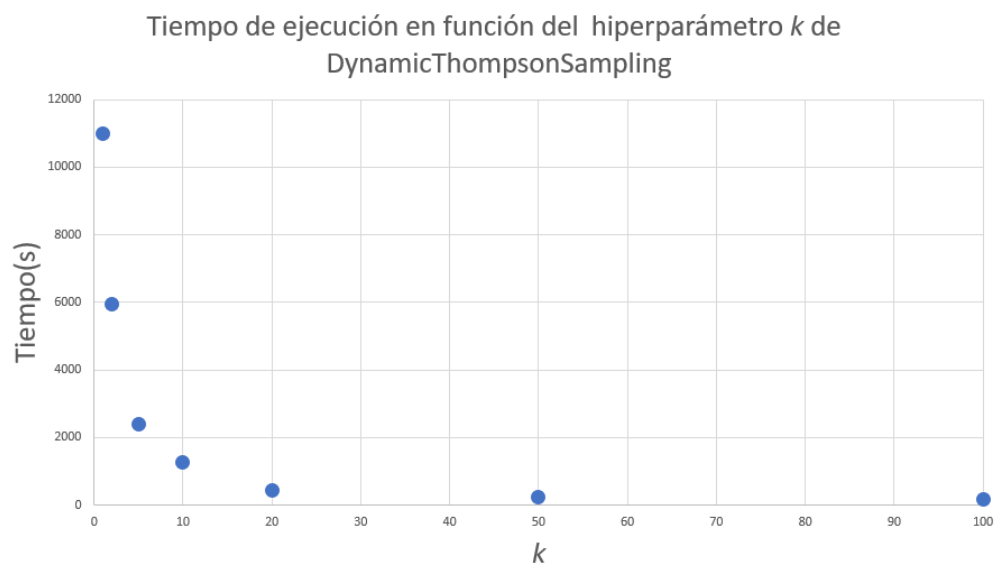


Figura 4.7: Gráfica de distintos valores del hiperparámetro k frente al tiempo de ejecución en Dynamic Thompson Sampling.

y $\beta_e(i)$ aumentan hay un ítem que pasa a ser el favorito. Si este ítem lo mantenemos demasiado tiempo (k alta) es muy probable que se obtengan resultados no óptimos. Al no permitir Dynamic Thompson Sampling que las curvas de la distribución Beta se cierren demasiado, la elección del brazo cada iteración tiene un poco más de variabilidad y por tanto ya no es solo un ítem el favorito si no un grupo de ítems de los que hay probabilidades no despreciables de elegir. Ya sabemos que Dynamic Thompson Sampling rinde un poco mejor que Thompson Sampling así que tener un grupo de favoritos frente a un solo favorito ya de por sí genera buenos resultados, pero además permite aumentar este valor k más que en Thompson Sampling. En la figura 4.8 tenemos las gráficas de la evolución del acierto y la novedad a lo largo de la ejecución. Podemos observar como para $k = 10$ el valor del acierto es máximo. Incluso $k = 20$ obtiene mejores resultados de acierto que $k = 1$ tardando aproximadamente 20 veces menos tiempo en ejecutarse.

4.2. Comparativa de algoritmos

Este último experimento consiste en comparar todos los algoritmos entre ellos para observar que tal rinden en cada una de las métricas. De esta manera podremos tener una imagen general de que algoritmos rinden mejor que otros en que métricas. En este caso se ha utilizado la partición de test de la base de datos que es una división aleatoria del 80 % del total de Movielens 1M [Movielens, 1997].

Debido a que el número de algoritmos que se van a comparar es elevado, se divide esta sección en distintos apartados donde se compararán los algoritmos basados en algún algoritmo original. Para ello se proporcionan una tabla que muestran la media y la desviación estándar de los valores en cinco

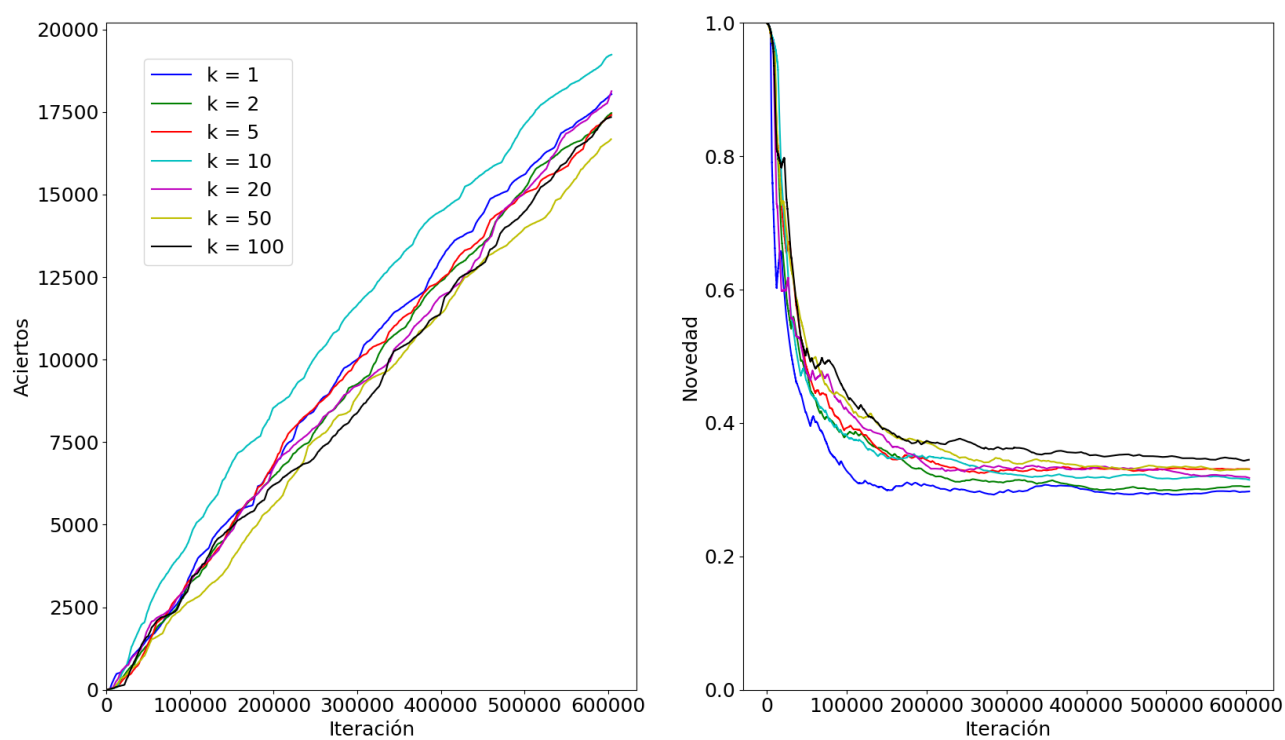
Dynamic Thompson Sampling para distintos valores de k 

Figura 4.8: Gráfica de acierto y novedad para distintos valores del hiperparámetro k en Dynamic Thompson Sampling.

ejecuciones donde el nombre del algoritmo con mayor acierto estará en negrita además del mejor valor de cada métrica. Finalmente se presenta una comparativa de los resultados de los algoritmos que hayan obtenido mayor acierto en su categoría.

4.2.1. Comparativa de algoritmos basados en ϵ -greedy

| | Adap ϵ -Greedy | ϵ -first Greedy | ϵ -Greedy | VDBE ϵ -Greedy |
|-----------|-------------------------------------|-------------------------------------|--------------------|-------------------------------------|
| ILD@10 | 0.72 ± 0.018 | 0.773 ± 0.038 | 0.73 ± 0.045 | 0.744 ± 0.042 |
| Unex@10 | 0.818 ± 0.08 | 0.843 ± 0.013 | 0.817 ± 0.028 | 0.817 ± 0.029 |
| Novedad | 0.253 ± 0.029 | 0.231 ± 0.003 | 0.229 ± 0.003 | 0.23 ± 0.007 |
| Gini | 0.131 ± 0.008 | 0.12 ± 0.006 | 0.121 ± 0.007 | 0.139 ± 0.008 |
| Acierto | 0.104 ± 0.004 | 0.101 ± 0.005 | 0.097 ± 0.006 | 0.112 ± 0.007 |
| Tiempo(s) | 502 ± 44 | 472 ± 24 | 540 ± 134 | 485 ± 26 |

Tabla 4.1: Tabla comparativa de los algoritmos basados en ϵ -greedy. Se muestra la media y la desviación típica de cinco ejecuciones. El valor en negrita indica el mejor valor de la métrica. Los hiperparámetros utilizados son: $limit = 1000$ y $f = 7$ para Adaptive ϵ -Greedy, $iterations = 100000$ para ϵ -first Greedy, $\epsilon = 0,1$ para ϵ -Greedy y $\sigma = 0,04$ y $\delta = 0,001$ para VDBE ϵ -Greedy.

ϵ -first Greedy obtiene los mejores valores en las métricas de diversidad, en el coeficiente de Gini y en tiempo de ejecución. Sin embargo, es VDBE ϵ -Greedy el que mayor acierto tiene. La diferencia de aciertos parece baja, pero es una mejora del 10% y por tanto notable. VBDE ϵ -Greedy no da malos resultados en el resto de las métricas y siendo el que mejor acierto tiene es el algoritmo que se va a utilizar en la comparativa final. Cabe destacar que en todas las métricas hay alguna de las variantes que rinde mejor que ϵ -Greedy y que además todas obtienen un acierto mejor. Esto es debido a que cualquiera de estas variaciones tiene la posibilidad de comportarse de manera estática y por tanto como mínimo deben rendir mejor que ϵ -Greedy que no tiene margen de mejora respecto a la exploración estática.

4.2.2. Comparativa de algoritmos basados en UCB

En este caso AdaptiveUCB obtiene mejor rendimiento que UCB en todas las métricas 4.2. Es especialmente llamativo la mejoría que supone esta modificación en el acierto con un aumento del 15%. El hecho de que el valor de γ dependa del número de aciertos y fallos en la manera que lo hace permite al algoritmo adaptarse. Es por esto que aumenta el acierto. Lo que es verdaderamente sorprendente es que esta variación rinde mejor que el algoritmo base en todas las métricas.

| | AdaptiveUCB | UCB |
|-----------|--------------------------------------|-------------------|
| ILD@10 | 0.762 \pm 0.018 | 0.751 \pm 0.044 |
| Unex@10 | 0.855 \pm 0.014 | 0.854 \pm 0.016 |
| Novedad | 0.363 \pm 0.01 | 0.352 \pm 0.028 |
| Gini | 0.117 \pm 0.005 | 0.124 \pm 0.005 |
| Acierto | 0.1019 \pm 0.005 | 0.088 \pm 0.005 |
| Tiempo(s) | 596 \pm 13 | 606 \pm 15 |

Tabla 4.2: Tabla comparativa de los algoritmos basados en UCB. Se muestra la media y la desviación típica de cinco ejecuciones. El valor en negrita indica el mejor valor de la métrica. Los hiperparámetros utilizados son: $\gamma = 0,01$ para UCB ya que AdaptiveUCB no dispone de ninguno.

| | DynThoSam | ThoSam |
|-----------|-------------------------------------|-------------------------------------|
| ILD@10 | 0.744 \pm 0.053 | 0.722 \pm 0.021 |
| Unex@10 | 0.828 \pm 0.013 | 0.824 \pm 0.013 |
| Novedad | 0.29 \pm 0.005 | 0.304 \pm 0.005 |
| Gini | 0.172 \pm 0.002 | 0.162 \pm 0.004 |
| Acierto | 0.131 \pm 0.002 | 0.122 \pm 0.003 |
| Tiempo(s) | 657 \pm 22 | 2992 \pm 390 |

Tabla 4.3: Tabla comparativa de los algoritmos basados en Thompson Sampling. Se muestra la media y la desviación típica de cinco ejecuciones. El valor en negrita indica el mejor valor de la métrica. Los hiperparámetros utilizados son: $\alpha_0 = 1$, $\beta_0 = 20$, $k = 10$ y $C = 100$ para Dynamic Thompson Sampling y $\alpha_0 = 1$, $\beta_0 = 100$ y $k = 2$ para Thompson Sampling.

4.2.3. Comparativa de algoritmos basados en Thompson Sampling

En este caso la variante dinámica también rinde mejor en términos de acierto. En las métricas de diversidad también rinde mejor, aunque en novedad e índice de Gini es un poco peor. Lo que es realmente notable es el tiempo de ejecución. Por lo que hemos visto en la sección 3.3.4 Dynamic Thompson Sampling nos permite aumentar más el valor del hiperparámetro k . En el caso del Thompson Sampling tenemos $k = 2$ por lo que el tiempo de ejecución es aproximadamente la mitad que si no se utilizara el hiperparámetro. En Dynamic Thompson Sampling $k = 10$ y por tanto tarda aproximadamente diez veces menos y de ahí la diferencia.

4.2.4. Comparativa de otros algoritmos

| | CLUB | LinUCB | EXP3 | Knn |
|-----------|-------------------------------------|--------------------------------|-------------------------------------|-------------------|
| ILD@10 | 0.372 ± 0.076 | 0.681 ± 0.045 | 0.757 ± 0.003 | 0.749 ± 0.007 |
| Unex@10 | 0.494 ± 0.036 | 0.762 ± 0.015 | 0.84 ± 0.002 | 0.831 ± 0.004 |
| Novedad | 0.596 ± 0.008 | 0.486 ± 0.005 | 0.557 ± 0.002 | 0.358 ± 0.004 |
| Gini | 0.109 ± 0.01 | 0.112 ± 0.001 | 0.111 ± 0.001 | 0.123 ± 0.004 |
| Acierto | 0.06 ± 0.005 | 0.068 ± 0.001 | 0.091 ± 0.000 | 0.089 ± 0.002 |
| Tiempo(s) | 12941 ± 1907 | 224 ± 14 | 1153 ± 55 | 2816 ± 420 |

Tabla 4.4: Tabla comparativa del resto de algoritmos. Se muestra la media y la desviación típica de cinco ejecuciones. El valor en negrita indica el mejor valor de la métrica. Los hiperparámetros utilizados son: $\alpha = 0,2$ y $\alpha_2 = 0,3$ para CLUB, $\alpha = 0,01$ para LinUCB, $\gamma = 0,6$ para EXP3 y $k = 10$ para knn.

En este caso no todos los algoritmos que se comparan siguen la misma filosofía. Es por esto que solamente se va a eliminar uno de los algoritmos contextuales (CLUB y LinUCB) y los otros dos (EXP3 y Knn) se utilizan en la comparativa final como *baseline* de bandidos multibrazo adversarios y filtrado colaborativo respectivamente.

Podemos observar en 4.4 como el rendimiento general de CLUB es muy malo. Esto puede ser debido a que la base de datos con la que estamos trabajando no dispone de mucha descripción del contenido (solamente 14 géneros de películas) y por tanto la agrupación que se hace internamente en el algoritmo no genera el efecto deseado. Por otro lado, LinUCB, teniendo una aproximación parecida, rinde mucho mejor, especialmente en tiempo de ejecución y en métricas de diversidad.

EXP3 es un algoritmo de bandidos multibrazo adversarios que como podemos ver rinde muy bien en la mayoría de las métricas y destaca en la novedad respecto a otros bandidos multibrazo. Por último, Knn es el *baseline* que se utiliza para comparar. Se trata de la típica aproximación que se utiliza actualmente en los sistemas de recomendación y por tanto es interesante ver cómo se comportan los nuevos algoritmos respecto al estado del arte.

4.2.5. Comparativa final de algoritmos

En 4.5 la recomendación aleatoria se utiliza como *baseline*. Es el más rápido de los algoritmos ya que no necesita hacer ningún cálculo, solamente elegir aleatoriamente de una lista. Además, su novedad y coeficiente de Gini son los mejores valores y los podemos utilizar para comparar con el resto de los algoritmos. Sin embargo, la recomendación aleatoria no es una estrategia que tenga sentido utilizar y es por esto que el acierto es muy bajo. Cabe destacar que no obtiene los mejores valores en diversidad.

Por otro lado, el algoritmo con mayor acierto es Dynamic Thompson Sampling. Su acierto es considerablemente mayor al resto de algoritmos, pero se puede ver como se ve perjudicado en métricas de diversidad, novedad y coeficiente de Gini. Este comportamiento hasta cierto punto tiene sentido, si quieres acertar más, es muy probable que debas sacrificar otras métricas. El tiempo de ejecución de este algoritmo no es tan elevado como podría llegar a ser gracias al hiperparámetro k presentado en este trabajo.

VDBE ϵ -Greedy tiene un comportamiento parecido a Dynamic Thompson Sampling, obtiene un acierto considerablemente alto, pero sufre en métricas de diversidad, especialmente novedad y en el coeficiente de Gini. Sin embargo, ya hemos visto que supone una mejora considerable respecto al ϵ -Greedy original en acierto. Su novedad es la más baja de todos los algoritmos.

Respecto al resto de algoritmos, LinUCB rinde peor. Sus métricas de diversidad son las peores. Su novedad y coeficiente de Gini son notablemente mejores pero su acierto es escaso. Esto puede ser debido a que el número de géneros de películas de la base de datos es muy pequeño y por tanto la información del contexto no es suficiente. Su tiempo de ejecución es bueno, pero consiguiendo tan poco acierto es probable que no merezca la pena.

Knn y EXP3 tienen un comportamiento parecido, no destacan en ninguna métrica, pero consiguen valores medios y altos en casi todas. Knn es el algoritmo más lento de todos y no parece merecer la pena observando los valores que tiene en el resto de las métricas. EXP3 por otro lado, obtiene una novedad notoriamente alta (mucho mayor al resto de bandidos multibrazo) y una diversidad y coeficiente de Gini buenos con un acierto aceptable.

AdaptiveUCB es el algoritmo con mayor diversidad, su acierto es relativamente alto y su novedad y coeficiente de Gini son medios. Si lo comparamos con Dynamic Thompson Sampling o VDBE ϵ -Greedy vemos que su acierto es peor pero que obtiene un rendimiento mucho más consistente en el resto de las métricas.

Todos estos comportamientos surgen de la optimización de hiperparámetros realizada previamente donde se han escogido los valores que maximizaran el acierto. Cabe mencionar que la mayoría de estos algoritmos con otros valores en sus hiperparámetros pueden llegar a sacrificar ciertas métricas por otras. Es decir, Dynamic Thompson Sampling obtiene un acierto enorme pero una novedad baja,

con otra configuración de los hiperparámetros podría ser posible aumentar la novedad sacrificando por ejemplo acierto. Sin embargo, la métrica más importante es el acierto y por eso se ha seguido este criterio.

Las gráficas 4.9 y 4.10 se utilizan para ver la evolución de estas métricas a lo largo de una ejecución. Hay algoritmos que pueden tener muy buen rendimiento en el comienzo de la ejecución, pero luego caer (por ejemplo, LinUCB en acierto) y puede ser interesante observar estos comportamientos.

| | AdapUCB | DynThoSam | EXP3 | Knn | LinUCB | Random | VDBE |
|-----------|--------------|--------------|-------|-------|--------|--------------|-------|
| ILD@10 | 0.762 | 0.744 | 0.757 | 0.749 | 0.681 | 0.748 | 0.744 |
| Unex@10 | 0.855 | 0.828 | 0.84 | 0.831 | 0.762 | 0.831 | 0.817 |
| Novedad | 0.363 | 0.29 | 0.557 | 0.358 | 0.486 | 0.713 | 0.23 |
| Gini | 0.117 | 0.172 | 0.111 | 0.123 | 0.112 | 0.024 | 0.139 |
| Acierto | 0.1019 | 0.131 | 0.091 | 0.089 | 0.068 | 0.021 | 0.112 |
| Tiempo(s) | 596 | 657 | 1153 | 2816 | 224 | 44 | 485 |

Tabla 4.5: Tabla comparativa de cada uno de los mejores algoritmos de los apartados anteriores además de algunos *baseline*. Cuanto más oscura la celda mejor es el valor de esa métrica para ese algoritmo y el valor en negrita indica el mejor valor de la métrica. Los hiperparámetros utilizados son: $\alpha_0 = 1$, $\beta_0 = 20$, $k = 10$ y $C = 100$ para Dynamic Thompson Sampling, $\gamma = 0,6$ para EXP3, $k = 10$ para Knn, $\alpha = 0,01$ para LinUCB y $\sigma = 0,04$ y $\delta = 0,001$ para VDBE ϵ -Greedy. Tanto AdaptiveUCB como Random no disponen de hiperparámetros.

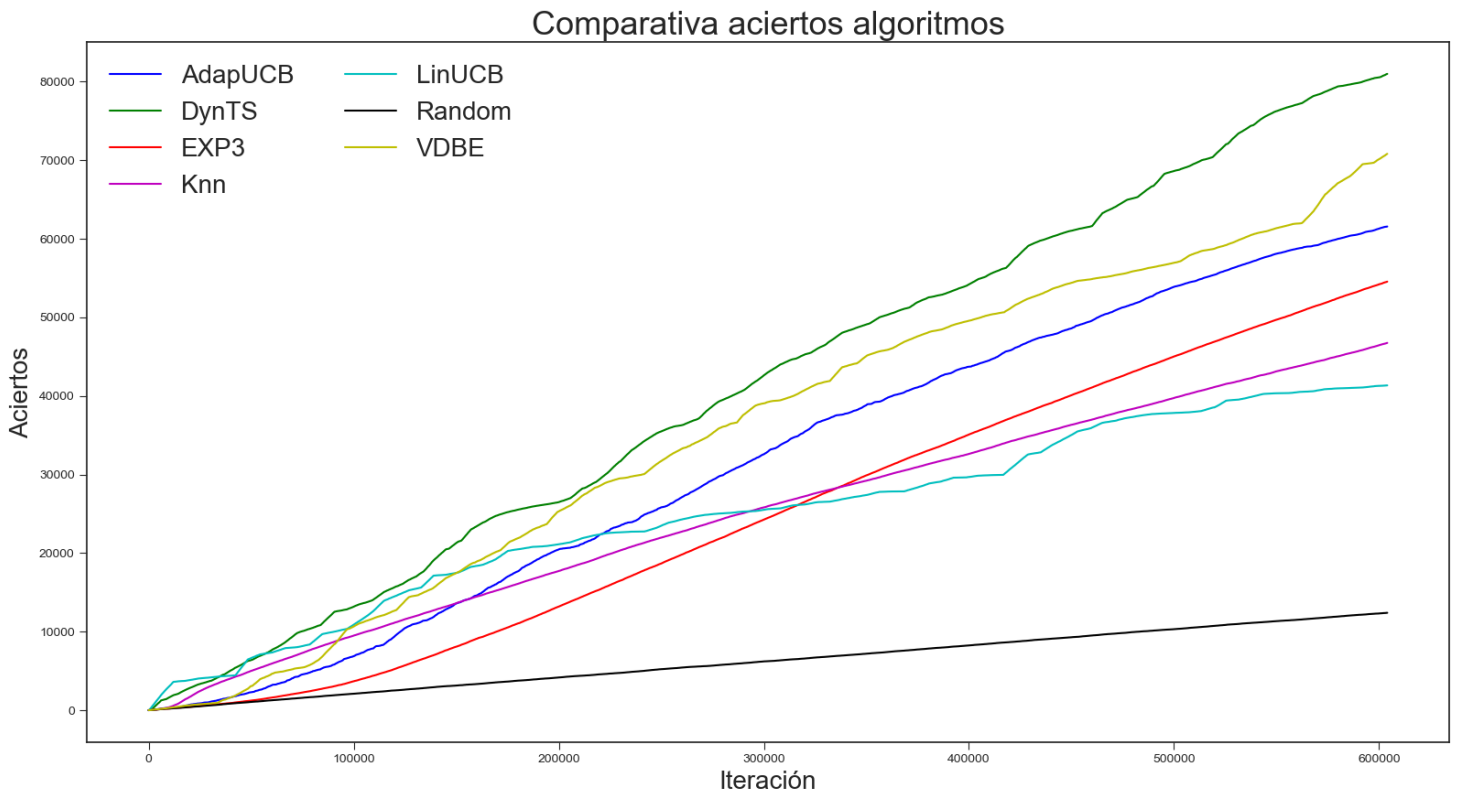


Figura 4.9: Evolución del número de aciertos a lo largo de una ejecución para los algoritmos optimizados. Los hiperparámetros utilizados son: $\alpha_0 = 1$, $\beta_0 = 20$, $k = 10$ y $C = 100$ para Dynamic Thompson Sampling, $\gamma = 0,6$ para EXP3, $k = 10$ para Knn, $\alpha = 0,01$ para LinUCB y $\sigma = 0,04$ y $\delta = 0,001$ para VDBE ϵ -Greedy. Tanto AdaptiveUCB como Random no disponen de hiperparámetros.

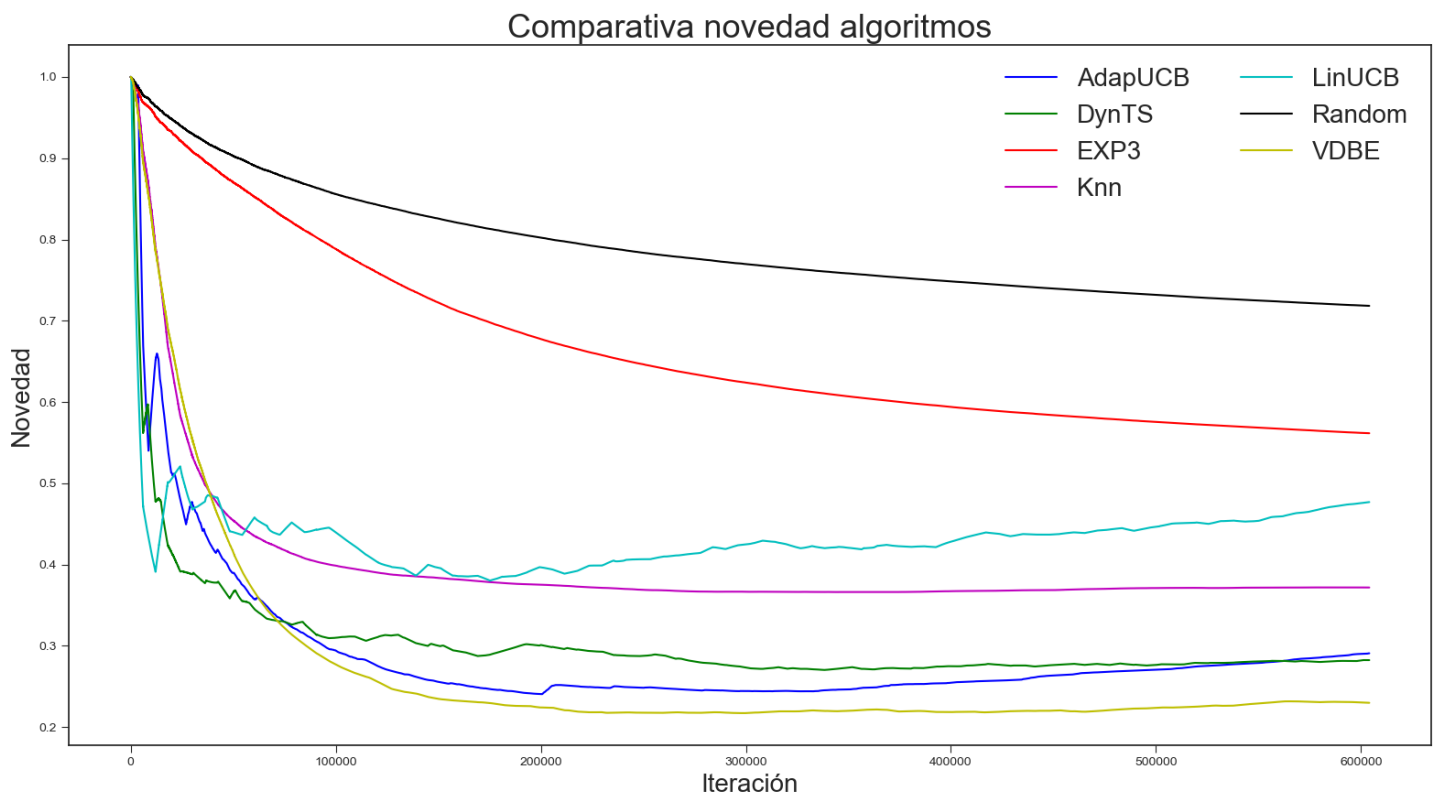


Figura 4.10: Evolución de la novedad a lo largo de una ejecución para los algoritmos optimizados.

Los hiperparámetros utilizados son: $\alpha_0 = 1$, $\beta_0 = 20$, $k = 10$ y $C = 100$ para Dynamic Thompson Sampling, $\gamma = 0,6$ para EXP3, $k = 10$ para Knn, $\alpha = 0,01$ para LinUCB y $\sigma = 0,04$ y $\delta = 0,001$ para VDBE ϵ -Greedy. Tanto AdaptiveUCB como Random no disponen de hiperparámetros.

CONCLUSIONES Y TRABAJO FUTURO

En esta sección se presentan las conclusiones del trabajo 5.1 y el trabajo futuro y posibles ampliaciones del mismo 5.2.

5.1. Conclusión

En este trabajo se ha expandido el estudio que se realizó en trabajos anteriores sobre los algoritmos de bandidos multibrazo en recomendación. En este caso se ha indagado en la exploración dinámica de los mismos. Para tener una *baseline* sólida con la que trabajar también se ha hecho uso de un algoritmo de filtrado colaborativo (knn), dos bandidos multibrazo basados en contexto (LinUCB y CLUB), uno basado en adversarios (EXP3) y los bandidos multibrazo de exploración estática implementados en trabajos anteriores (ϵ -greedy, Upper Confidence Bound y Thompson Sampling). Para evaluar los algoritmos las métricas que se han utilizado han sido: cierto, novedad, *intra-list diversity*, *unexpectedness*, coeficiente de Gini y tiempo de ejecución.

Hay que darse cuenta de que cuando añades la exploración dinámica a uno de estos algoritmos el rendimiento del algoritmo no puede empeorar. Esto es debido a que como mínimo el aspecto dinámico puede elegir mantenerse estático y por tanto tendrá un rendimiento igual. Es por esto por lo que la parte de búsqueda de hiperparámetros es importante. Si tomamos como *baseline* el rendimiento de la variante estática del algoritmo, necesitaremos encontrar una configuración de los hiperparámetros que la supere. Aunque aumentemos el acierto, puede ocurrir que otras dimensiones de la recomendación se vean afectadas negativamente y es por esto por lo que se utilizan otras métricas para evaluar. Es por todo esto que en los resultados finales observamos como todos los algoritmos implementados en este trabajo superan a los anteriores en términos de acierto.

El experimento final donde se comparan todos los algoritmos nos lleva a ciertas conclusiones. Lo primero es que no hay ningún algoritmo que sea el mejor en todos los aspectos. Cada uno de ellos sacrifica rendimiento en ciertos aspectos para maximizar otros. Sin embargo, si se puede ver como hay algún algoritmo que en general rinde mejor que otros. Por otro lado, tenemos claros ganadores en determinadas métricas. Por ejemplo, Dynamic Thompson Sampling obtiene un acierto muy elevado

respecto al resto de algoritmos y gracias al hiperparámetro k presentado en este trabajo, si tiempo de ejecución no es tan elevado como podría ser. También destaca AdaptiveUCB por tener una diversidad alta aun manteniendo un acierto alto.

Por la naturaleza impredecible de los seres humanos, no es posible tener un sistema de recomendación perfecto hoy día, pero la investigación de nuevas técnicas permite que los sistemas evolucionen y cada vez realicen mejor su tarea.

5.2. Trabajo futuro

Además de Movielens 1M existen otras bases de datos que pueden ser utilizadas para realizar estos experimentos o parecidos. Se pueden utilizar otras bases de datos para ver cómo se comportan los algoritmos en distintos escenarios o simplemente aumentar el tamaño de la base de datos utilizando, por ejemplo, Movielens 20M.

La exploración dinámica es un aspecto de los bandidos multibrazo aún poco explorada. Este trabajo escoge algunos algoritmos ya creados además de crear uno nuevo y modificar otro pero la cantidad de variaciones que puede haber es enorme. Profundizar en la exploración dinámica puede llevar a mejoras notables en el rendimiento. Las métricas utilizadas en este trabajo para medir el rendimiento son limitadas. Añadir o modificar alguna métrica puede ser interesante si se quiere medir nuevas dimensiones.

BIBLIOGRAFÍA

- [Aróstegui and Castells, 2020] Aróstegui, J. and Castells, P. (2020). Novedad y diversidad en recomendación con bandidos multi-brazo. *UAM. Departamento de Ingeniería Informática*.
- [Auer et al., 2002a] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002a). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256.
- [Auer et al., 2002b] Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. (2002b). The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77.
- [Balabanovic and Shoham, 1997] Balabanovic, M. and Shoham, Y. (1997). Fab: Content-based, collaborative recommendation. *Communications of the ACM*, 40:66–72.
- [Bertsekas and Tsitsiklis, 1995] Bertsekas, D. P. and Tsitsiklis, J. N. (1995). Neuro-dynamic programming: an overview. In *Proceedings of 1995 34th IEEE conference on decision and control*, volume 1, pages 560–564. IEEE.
- [Borràs et al., 2014] Borràs, J., Moreno, A., and Valls, A. (2014). Intelligent tourism recommender systems: A survey. *Expert Systems with Applications*, 41:7370–7389.
- [Burke, 2007] Burke, R. (2007). *Hybrid Web Recommender Systems*, pages 377–408. Springer Berlin Heidelberg, Berlin, Heidelberg.
- [Caelen and Bontempi, 2008] Caelen, O. and Bontempi, G. (2008). Improving the exploration strategy in bandit algorithms. In Maniezzo, V., Battiti, R., and Watson, J.-P., editors, *Learning and Intelligent Optimization*, pages 56–68, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Castells et al., 2015] Castells, P., Hurley, N. J., and Vargas, S. (2015). *Novelty and Diversity in Recommender Systems*, pages 881–918. Springer.
- [Castells and Sanz-Cruzado, 2019] Castells, P. and Sanz-Cruzado, J. (2019). **kNNBandit**.
- [Chapelle and Li, 2011] Chapelle, O. and Li, L. (2011). An empirical evaluation of thompson sampling. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 2249–2257. Curran Associates, Inc.
- [dos Santos Mignon and de Azevedo da Rocha, 2017] dos Santos Mignon, A. and de Azevedo da Rocha, R. L. (2017). An adaptive implementation of ϵ -greedy in reinforcement learning. *Procedia Computer Science*, 109:1146–1151. 8th International Conference on Ambient Systems, Networks and Technologies, ANT-2017 and the 7th International Conference on Sustainable Energy Information Technology, SEIT 2017, 16-19 May 2017, Madeira, Portugal.
- [Ek and Strigsson, 2015] Ek, F. and Strigsson, R. (2015). Recommender systems; contextual multiarmed bandit algorithms for the purpose of targeted advertisements within e-commerce.
- [Ekstrand et al., 2011] Ekstrand, M. D., Riedl, J. T., and Konstan, J. A. (2011). *Collaborative filtering recommender systems*. Now Publishers Inc.

- [Even-Dar et al., 2002] Even-Dar, E., Mannor, S., and Mansour, Y. (2002). Pac bounds for multi-armed bandit and markov decision processes. In Kivinen, J. and Sloan, R. H., editors, *Computational Learning Theory*, pages 255–270, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Figueroa et al., 2004] Figueroa, C. G., Alonso, J. L., Zazo, A. F., and Rodríguez, E. (2004). Algunas técnicas de clasificación automática de documentos.
- [Galbraith, 2016] Galbraith, B. (2016). [Python library for Multi-Armed Bandits](#).
- [Gentile et al., 2014] Gentile, C., Li, S., and Zappella, G. (2014). Online clustering of bandits. In *International Conference on Machine Learning*, pages 757–765. PMLR.
- [Goldberg et al., 1992] Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70.
- [GroupLens, 1997] GroupLens (1997). .
- [Gupta et al., 2011] Gupta, N., Granmo, O.-C., and Agrawala, A. (2011). Thompson sampling for dynamic multi-armed bandits. In *2011 10th International Conference on Machine Learning and Applications and Workshops*, volume 1, pages 484–489. IEEE.
- [Hao et al., 2020] Hao, B., Lattimore, T., and Szepesvari, C. (2020). Adaptive exploration in linear contextual bandit. In *International Conference on Artificial Intelligence and Statistics*, pages 3536–3545. PMLR.
- [Konstan et al., 1997] Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Riedl, J. (1997). Grouplens: Applying collaborative filtering to usenet news. *Commun. ACM*, 40(3):77–87.
- [Li et al., 2010] Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670.
- [Mehrotra et al., 2020] Mehrotra, R., Xue, N., and Lalmas, M. (2020). Bandit based optimization of multiple objectives on a music streaming platform. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3224–3233.
- [Movielens, 1997] Movielens (1997). [Movielens 1M](#).
- [Ning et al., 2015] Ning, X., Desrosiers, C., and Karypis, G. (2015). *A Comprehensive Survey of Neighborhood-Based Recommendation Methods*, pages 37–76. Springer US, Boston, MA.
- [Real and Vargas, 1996] Real, R. and Vargas, J. (1996). The probabilistic basis of jaccard’s index of similarity. *Systematic Biology - SYST BIOL*, 45:380–385.
- [Ricci et al., 2010] Ricci, F., Rokach, L., and Shapira, B. (2010). *Recommender Systems Handbook*, volume 1-35, pages 37–353.
- [Shardanand and Maes, 1995] Shardanand, U. and Maes, P. (1995). Social information filtering: Algorithms for automating “word of mouth”. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI ’95, page 210–217, USA. ACM Press/Addison-Wesley Publishing Co.
- [Sun and Li, 2020] Sun, L. and Li, K. (2020). Adaptive operator selection based on dynamic thompson sampling for MOEA/D. *CoRR*, abs/2004.10874.
- [Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduc-*

tion. The MIT Press, second edition.

- [Tokic, 2010] Tokic, M. (2010). Adaptive ϵ -greedy exploration in reinforcement learning based on value differences. In Dillmann, R., Beyerer, J., Hanebeck, U. D., and Schultz, T., editors, *KI 2010: Advances in Artificial Intelligence*, pages 203–210, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Vermorel and Mohri, 2005a] Vermorel, J. and Mohri, M. (2005a). Multi-armed bandit algorithms and empirical evaluation. In *European conference on machine learning*, pages 437–448. Springer.
- [Vermorel and Mohri, 2005b] Vermorel, J. and Mohri, M. (2005b). Multi-armed bandit algorithms and empirical evaluation. In Gama, J., Camacho, R., Brazdil, P. B., Jorge, A. M., and Torgo, L., editors, *Machine Learning: ECML 2005*, pages 437–448, Berlin, Heidelberg. Springer Berlin Heidelberg.

